QUEST SOFTWARE®

JProbe® 8.3

Tutorials

© 2009 Quest Software, Inc. ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

If you have any questions regarding your potential use of this material, contact:
Quest Software World Headquarters
LEGAL Dept
5 Polaris Way
Aliso Viejo, CA 92656
www.quest.com
email: legal@quest.com

Refer to our Web site for regional and international office information.

JProbe Patents

Patent pending.

Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BusinessInsight, ChangeAuditor, ChangeManager, DeployDirector, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, ERDisk, Foglight, GPOADmin, Imceda, IntelliProfile, InTrust, Invirtus, iToken, I/Watch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL LiteSpeed, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer Pro, vPackager, vRanger, vRanger Pro, vSpotlight, vStream, vToad, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, Vizioncore vTraffic, Vizioncore vWorkflow, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Disclaimer

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST'S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

License Credits and Third Party Information

See *Third_Party_Contributions.htm* in your JProbe *\doc* installation directory.

Tutorials
November 2009

Table of Contents

Introduction to This Guide 7

About JProbe 8

About This Guide 8

JProbe Documentation Suite 9

Core Documentation Set 9

Feedback on the Documentation 10

Text Conventions 10

About Quest Software, Inc. 10

Contacting Quest Software 11

Contacting Quest Support 11

Quest Communities 12

Memory Analysis Demos 13

Summary of Demos for Memory 14

LeakExample Tutorial 16

Step 1: Setting Up the Memory Leak Session 17

Step 2: Running the Memory Leak Session 20

Step 3: Identifying Loitering Objects 21

Step 4: Investigating Loitering Objects 22

Step 5: Running the Memory Leak Session with Improved Code 26

Network Tutorial 28

Step 1: Setting Up the Network Session 29

Step 2: Running the Network Session 30

Step 3: Identifying Large Allocations of Short-Lived Objects 31

Step 4: Investigating Large Allocations of Short-Lived Objects 32

Step 5: Running the Network Session with Improved Code 34

Performance Analysis Demos 39
Summary of Demos for Performance 40
Philosopher's Diner Tutorial 42
Step 1: Setting Up the Diner Session 42
Step 2: Running the Diner Session 43
Step 3: Investigating the Deadlock 45
Polynomial Tutorial 47
Step 1: Setting Up the Polynomial Session 48
Step 2: Running the Polynomial Session 51
Step 3: Identifying and Investigating the Performance Bottleneck 52
Step 4: Running the Polynomial with Improved Code 53
Step 5: Measuring the Performance Improvement 55

Coverage Analysis Demos 57
Summary of Demos for Coverage 58
Adventure Tutorial 59
Step 1: Setting Your Global Options 59
Step 2: Setting Up the Session for the First Test Case 60
Step 3: Running the First Test Case 61
Step 4: Setting Up and Running the Second Test Case 62
Step 5: Merging the Test Case Results 63
Step 6: Assessing Your Test Case Coverage 64

JProbe Game Pack for JavaEE 67
Getting Started with the Game Pack Demo 68
Installing the Game Pack Demo 68
System Requirements 69
Setting Environment Variables 69
Quick Game Pack Evaluation 70
Game Pack Memory Analysis Tutorials 71
Game Pack Performance Analysis Tutorial 83
Game Pack Coverage Analysis Tutorial 89
Advanced Game Pack Evaluation 95
Deploying the Game Pack Demo on Your Application Server 95
Running Game Pack with the JProbe Application 109

Game Pack Memory Analysis Tutorials 110
Game Pack Performance Analysis Tutorial....122
Game Pack Coverage Analysis Tutorial....128
Creating a User Account for Game Pack....134
Game Pack Known Issues....135

JProbe Treasurehunt....137
Installing the Treasurehunt Demo....138
Using the Treasurehunt Demo....138

Index....139

Introduction to This Guide

This chapter provides information about what is contained in the *JProbe Tutorials*. It also provides information about the JProbe documentation suite and Quest Software.

This chapter contains the following sections:

About JProbe....8
About This Guide....8
JProbe Documentation Suite....9
Text Conventions....10
About Quest Software, Inc.....10

About JProbe

JProbe is an enterprise-class Java profiler that provides intelligent diagnostics on memory usage, performance, and test coverage. It allows developers to quickly pinpoint and repair the root cause of application code performance and stability problems that obstruct component and integration integrity.

JProbe provides three types of analysis:

- Memory analysis—allows a developer to identify and resolve Java memory leaks and object cycling, to ensure optimal program efficiency and stability.
- Performance analysis—allows a developer to identify and resolve Java bottlenecks and deadlocks, to ensure optimal program performance and scalability.
- Coverage analysis—allows a developer to identify un-executed lines of code during unit testing, to ensure test coverage and program correctness.

JProbe also offers an Eclipse plugin that provides intelligent code performance analysis and problem resolution from within the Eclipse Java IDE.

About This Guide

The *JProbe Tutorials* contains a summary of all the demo applications that ship with JProbe. It also contains tutorials for some of these applications.

This document is intended for Java developers who want to learn how to configure JProbe to work with their application and run a JProbe analysis.

The *JProbe Tutorials* is organized as follows:

Chapter 1, Memory Analysis Demos—provides a summary of the Memory demo applications that ship with JProbe, and contains tutorials for some of these applications.

Chapter 2, Performance Analysis Demos—provides a summary of the Performance demo applications that ship with JProbe, and contains tutorials for some of these applications.

Chapter 3, Coverage Analysis Demos—provides a summary of the Coverage demo applications that ship with JProbe, and a tutorial for one of these applications.

Chapter 4, JProbe Game Pack for JavaEE—describes how to deploy the JProbe Game Pack demo software and presents how to use JProbe with JavaEE applications

that require a third-party application server. The Game Pack tutorials describe how to find two different kinds of memory leaks with the Memory analysis tool, how to investigate a performance bottleneck with the Performance analysis tool, and how to assess code coverage with the Coverage analysis tool.

Chapter 5, JProbe Treasurehunt—provides information about the Treasurehunt demo software that ships with JProbe.

JProbe Documentation Suite

The JProbe documentation suite is provided in a combination of online help, PDF, HTML, and TXT.

- **Online Help:** You can open the online help by clicking the **Help** icon on the JProbe toolbar.
- **PDF:** The complete JProbe documentation set is available in PDF format on SupportLink. The PDF documentation can also be found in the Documentation folder on the JProbe DVD. The default location of the documentation after an installation is *<jprobe_home>/docs*. Adobe Reader is required.
- **HTML:** *Release Notes* are provided in HTML and TXT format. The default location of this document after an installation is *<jprobe_home>/docs*.

  The *Ant Tasks User Manual* is also provided in HTML format. The default location of this document after an installation is *<jprobe_home>/automation/doc*. To open it, navigate to *index.html*.

Core Documentation Set

The core documentation set consists of the following files:

- *JProbe Installation Guide* (PDF)
- *JProbe User Guide* (PDF and online help)
- *JProbe Reference Guide* (PDF)
- *JProbe Plugins for Eclipse Guide* (PDF)
- *JProbe Tutorials* (PDF and online help)
- *JProbe Release Notes* (HTML and TXT)
- *Ant Tasks User Manual* (HTML)

Feedback on the Documentation

We are interested in receiving feedback from you about our documentation. For example, did you notice any errors in the documentation? Were any features undocumented? Do you have any suggestions on how we can improve the documentation? All comments are welcome. Please submit your feedback to the following email address:

am.docfeedback@quest.com

Please do not submit Technical Support related issues to this email address.

Text Conventions

The following table summarizes how text styles are used in this guide:

Convention	Description
`Code`	Monospace text represents code, code objects, and command-line input. This includes: • Java language source code and examples of file contents • Classes, objects, methods, properties, constants, and events • HTML documents, tags, and attributes
`Variables`	Monospace-plus-italic text represents variable code or command-line objects that are replaced by an actual value or parameter.
Interface	Bold text is used for interface options that you select (such as menu items) as well as keyboard commands.
Files, components, and documents	Italic text is used to highlight the following items: • Pathnames, file names, and programs • The names of other documents referenced in this guide

About Quest Software, Inc.

Now more than ever, organizations need to work smart and improve efficiency. Quest Software creates and supports smart systems management products—helping our

customers solve everyday IT challenges faster and easier. Visit *www.quest.com* for more information.

Contacting Quest Software

Email	*info@quest.com*
Mail	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	*www.quest.com*

Refer to our web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at: *http://support.quest.com*.

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the *Global Support Guide* for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: *http://support.quest.com/pdfs/Global Support Guide.pdf*.

Quest Communities

Get the latest product information, find helpful resources, and join a discussion with the JProbe Quest team and other community members. Join the JProbe community at: *http://jprobe.inside.quest.com/*.

1

Memory Analysis Demos

This chapter provides a summary of the Memory demo applications that ship with JProbe, and contains tutorials for some of these applications.

The source code and compiled classes for the Memory demos are located in the *<jprobe_home>/demos/memory* directory.

This chapter contains the following sections:

Summary of Demos for Memory 14
LeakExample Tutorial 16
Network Tutorial 28

Summary of Demos for Memory

The following table describes the purpose of the example applications.

Java SE Application	Purpose	More Information
Account.class	This application creates an account object and it is used by the *AccountInfo.class* application.	See the notes in *Account.java*.
AccountInfo.class	Displays three sets of account information to illustrate the impact of indirect object instantiation.	See the notes in *AccountInfo.java*.
LeakExample.class	This example illustrates how an obsolete collection reference may cause loitering objects. When the buttons are removed from a panel, the `JButton` objects are not removed from the Java heap.	See the notes in *LeakExample.java*. **Tutorial:** "LeakExample Tutorial" on page 16
LeakExample2.class	Similar to *LeakExample.class*, this application registers the buttons as listeners. The program demonstrates loitering objects caused by an obsolete listener.	See the notes in *LeakExample2.java*.
Network.class	This example simulates clients (threads) connecting to a server and querying a database. Temporary objects are created for the login data and for the connection.	See the notes in *Network.java*. **Tutorial:** "Network Tutorial" on page 28

Java SE Application	Purpose	More Information
Sim.class	This example simulates a network model in which a connection is made to verify the identity of the user. If the identity of the user is validated, the application extracts the desired data from the database and saves it in the result set.	See the notes in *Sim.java.*
StalledStack.class	This example shows how a stalled stack reference can hold objects in memory longer than necessary. You could be using the memory consumed by these objects for other tasks. Uses heap triggers.	See the notes in *StalledStack.java.*
Strings.class	Compares two algorithms: one that creates large allocations of string objects and one that does not.	See the notes in *Strings.java.*

JProbe also ships with a Java EE demo application called JProbe Game Pack. For more information, see “JProbe Game Pack for JavaEE” on page 67.

LeakExample Tutorial

The *LeakExample* program illustrates how an obsolete collection reference can anchor entire trees of loitering objects in the Java heap.

This tutorial demonstrates how to use JProbe to identify loitering objects in your code and how to reclaim memory by removing loitering `JButton` objects. The improved code shows a 89.15% improvement in how much memory is used by the instances allocated by *LeakExample* methods, and a 37.73% improvement in the overall memory used for the entire program.

Note **The values cited in this tutorial reflect the LeakExample running on Windows XP with Sun JDK delivered with JProbe**. You may see different values on your system, but the improvement in memory use should still be evident.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	*LeakExample.class*
Use Case:	Add buttons to a panel. Remove buttons from a panel.
Architecture:	Uses the class `JButton` to create buttons.
Hypothesis:	`JButton` objects are removed from the heap when the buttons are removed from the panel.

This tutorial assumes that you are running JProbe on your local machine. You can find more information about loitering objects in the *JProbe User Guide*.

This tutorial walks you through the following steps:

- Step 1: Setting Up the Memory Leak Session
- Step 2: Running the Memory Leak Session
- Step 3: Identifying Loitering Objects
- Step 4: Investigating Loitering Objects
- Step 5: Running the Memory Leak Session with Improved Code

Step 1: Setting Up the Memory Leak Session

In this step, you use the JProbe Console to create the session settings for this tutorial. The following procedures mention only the settings that you need to change or verify. If a setting is not mentioned, leave it blank or in its default state. The procedure assumes that you are running JProbe locally on your computer.

To set up the session:

1. In the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.

2. Click **Java Application > Java Application**.

3. Provide a name for this configuration in the Configuration Name text box (for example, *LeakExample*), then click **Next**.

   The Java Application tab is displayed.

4. In the Main Class area, select the **Execute a class** check box.

5. Click the browse button in the Main Class area and navigate to the *LeakExample.class* file in the *<jprobe_home>/demos/memory/leakexample* directory.

6. Click **OK**.

   The following information is displayed:

   - Main Class: `demos.memory.leakexample.LeakExample`
   - Working Directory: `<jprobe_home>`

7. Click the browse button beside the Classpath field.

8. In the Classpath dialog box, click **Add Working Directory**, then click **OK**.

   The working directory appears in the Classpath field.

9. Click **Next**.

   The Select a Java Virtual Machine tab is displayed.

10. If you want to change the default JVM, click the browse button beside the Java Executable field, select another JVM in the Java Virtual Machines dialog box, then click **OK**.

    Alternatively, you can use the JVM that is installed with JProbe to run with the tutorial. The java executable is *<jprobe_home>/bin/jre/bin/java.exe*.

11. Click **Next**.

The My Code tab is displayed.

12 In the Category/Program Name text box, type *LeakExampleDemo* (which specifies the name of the category in which you want to include your code), then click **Next**.

The Analysis Type tab is displayed.

13 Select the **Memory Analysis** option.

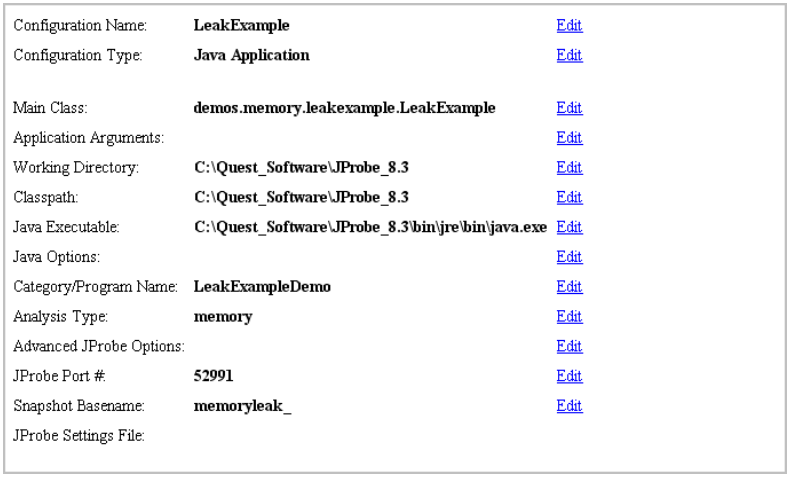
14 To disable the data recording at initial JVM start, select **Data Recording Off** on the **Initial Recording** tab.

15 Click **Next**.

The Advanced JProbe Options tab is displayed.

16 In the Snapshot Basename text box type *memoryleak_*, and then click **Next**.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.



Configuration Name:	**LeakExample**	Edit
Configuration Type:	**Java Application**	Edit
Main Class:	**demos.memory.leakexample.LeakExample**	Edit
Application Arguments:		Edit
Working Directory:	**C:\Quest_Software\JProbe_8.3**	Edit
Classpath:	**C:\Quest_Software\JProbe_8.3**	Edit
Java Executable:	**C:\Quest_Software\JProbe_8.3\bin\jre\bin\java.exe**	Edit
Java Options:		Edit
Category/Program Name:	**LeakExampleDemo**	Edit
Analysis Type:	**memory**	Edit
Advanced JProbe Options:		Edit
JProbe Port #:	**52991**	Edit
Snapshot Basename:	**memoryleak_**	Edit
JProbe Settings File:		

17 To create a JPL file and a startup script, click **Save & Integrate**.

18 Click **Save** to save the configuration file (*LeakExample_Mem_Settings.jpl*) into your working directory.

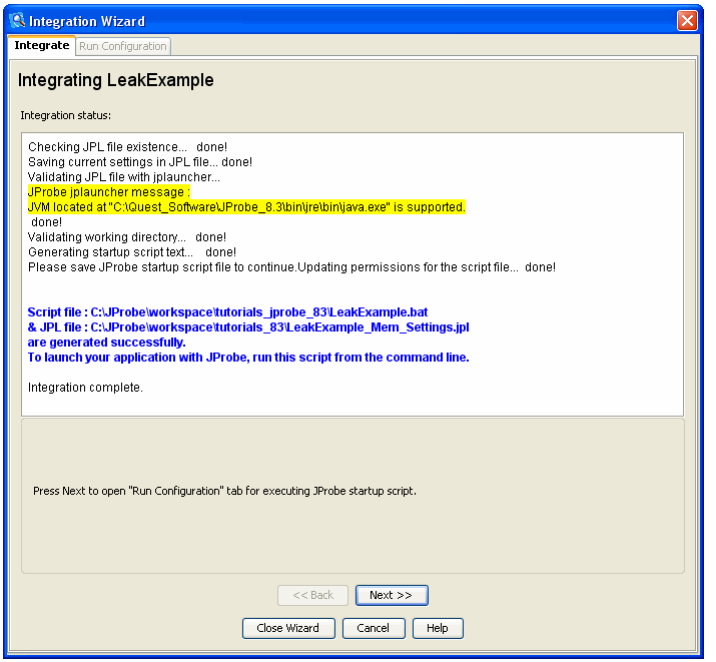
Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

19 In the Integrate tab, use the browse button to navigate to your working directory, and click **Save**.

Tip It is a good practice to save the script files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

JProbe validates the configuration file and creates a startup script file (for example, in Windows: *LeakExample.bat*, and in UNIX/Linux systems: *LeakExample.sh*).


Integration Wizard
Integrate
Run Configuration
Integrating LeakExample
Integration status:
Checking JPL file existence... done!
Saving current settings in JPL file... done!
Validating JPL file with jplauncher...
JProbe jplauncher message :
JVM located at "C:\Quest_Software\JProbe_8.3\bin\jre\bin\java.exe" is supported.
done!
Validating working directory... done!
Generating startup script text... done!
Please save JProbe startup script file to continue.Updating permissions for the script file... done!
Script file : C:\JProbe\workspace\tutorials_jprobe_83\LeakExample.bat
& JPL file : C:\JProbe\workspace\tutorials_83\LeakExample_Mem_Settings.jpl
are generated successfully.
To launch your application with JProbe, run this script from the command line.
Integration complete.
Press Next to open "Run Configuration" tab for executing JProbe startup script.
<< Back
Next >>
Close Wizard
Cancel
Help

20 When the integration is complete, click **Next**.

The Run Configuration tab is displayed.

21 Select the **Close syscon window on exit** check box to automatically close the *jpsyscon* window after the termination of the launched script, if the program exits normally.

22 To run your application with JProbe using the startup script, click **Run**.

Step 2: Running the Memory Leak Session

In this step, you exercise a use case on *LeakExample* that requires you to add buttons to a panel and then remove them. As the use case runs, you can see how many `JButton` objects have been added since the exercise started and how many remain in the heap when it ends.

Note This procedure assumes that the *LeakExample* program is already running (for instructions on how to execute the startup script, see step 22 in section "Step 1: Setting Up the Memory Leak Session" on page 17.

Alternatively, you can run the startup script from the command line:

In a Windows command window: `LeakExample.bat`

In a UNIX or Linux shell, from the directory where you saved the startup script:
`./LeakExample.sh`

To run the session:

1. Click **Attach to Session** on the JProbe Console toolbar.

   JProbe is searching for currently running engines and JVMs, and displays this list in the **Attach to Running Session** dialog box.

2. Select from the list the JProbe Analysis Engine running with the application and click **OK**.

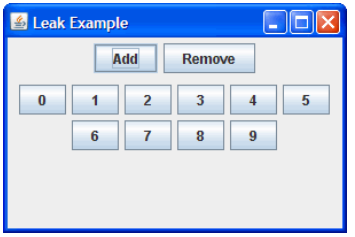
   After a few seconds the JProbe Console connects to the JProbe Analysis Engine, then the Runtime Summary view appears, with the Memory Pools tab on the foreground.

3. Click **Set Recording Level** on the toolbar.

4. In the Set Recording Level dialog box, select **Record Allocations and Stack Traces For All Instances**, then click **OK**.

5. In the Leak Example program window, click the **Add** button ten times.

   Ten buttons (numbered 0 to 9) are added on the program window.



6 Click the **Remove** button ten times.

The ten buttons are deleted from the program window.

7 Click **Set Recording Level** on the toolbar.

8 In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

9 Close the Leak Example program window.

JProbe disconnects from the running session. After a few seconds, the Instances view appears, displaying instances that were created during the use case.

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

Step 3: Identifying Loitering Objects

In this step, you look for loitering objects in the heap. Based on the hypothesis, you should expect the Recorded Count for the `JButton` class to be zero, because you removed all the buttons you added. In fact, the buttons are not removed and continue to loiter in the heap.

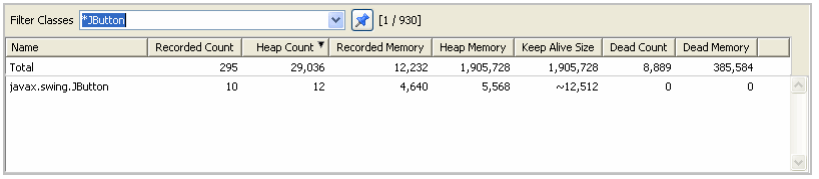
To identify loitering objects:

1 In the Instances view, select **Heap Count** from the Investigate by list.

2 Type `JButton` in the Filter Classes field and press **Enter** to locate the `JButton` class.

Note This field is case-sensitive.

The Instances list now displays only the `JButton` class.



The Recorded Count for `JButton` is 10, and the Dead Count is 0. This means that the buttons were removed from the Leak Example program window, but not from the Java heap.

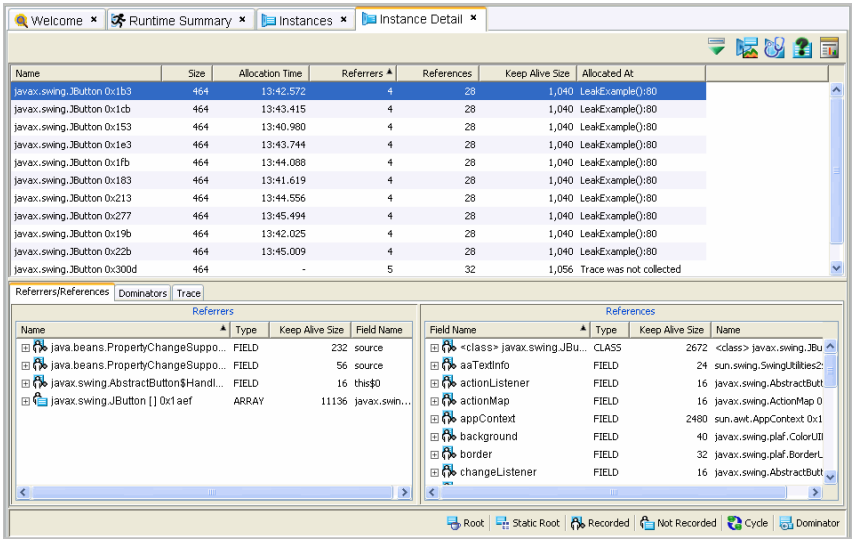
Step 4: Investigating Loitering Objects

In this step, you find the live object that continues to hold a reference to the loitering instances of `JButton`. From the Instances view, you can drill down to more detail on `JButton` in the Instance Detail view. Then you can open the Source view to see the code for the loiterer. You will find that the loitering `JButton` objects are being held by obsolete collection references from the array `JButton[]`.

To investigate loitering objects:

1. In the Instances view, select the `JButton` row and click **Instance Detail** on the toolbar.

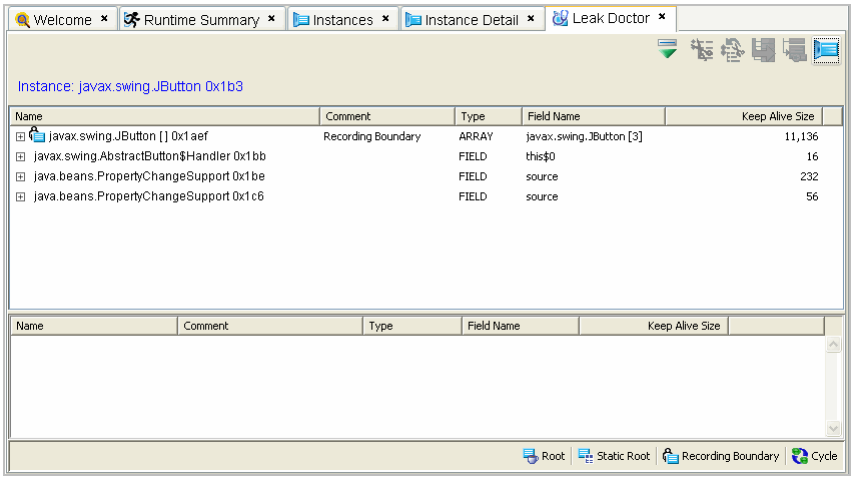
   The Instance Detail view appears, listing all the instances of `JButton` in the heap.



Tip The view displays 12 JButtons, not 10. The other two buttons were not added while the recording was ON, hence the recorded count in the Instances view is 10, and the heap count is 12.

2 In the upper table, click the first instance with an allocation time displayed, then click **Leak Doctor** on the toolbar.

The Leak Doctor view appears.



3 Select the first instance and click **Remove The Edge** on the toolbar.

A dialog box confirms that this instance is eligible to be garbage collected if you free the removed edge from your application code.

Instance Eligible for Garbage Collection
Congratulations, the instance can be garbage collected.
Your next step is to free the removed edges in your application code.
OK

4 Click **OK**.

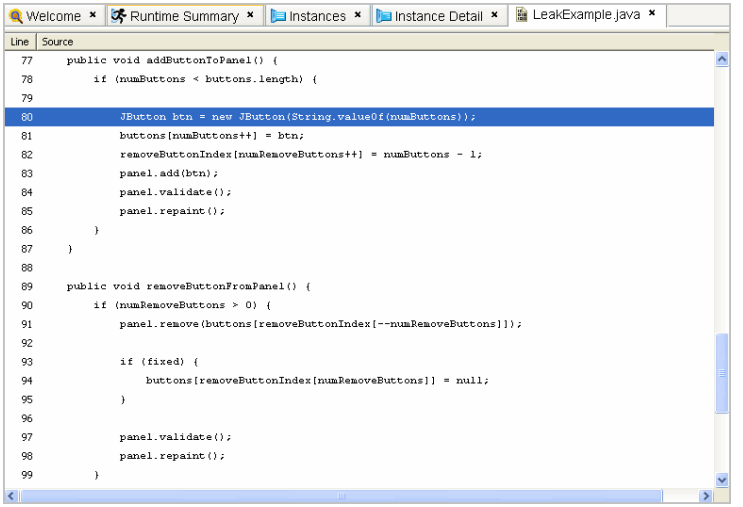
The selected instance moves from the upper table to the lower one. Your next step is to find the method that allocated the instance in order to understand why it is not being removed.

5 Close the Leak Doctor view by clicking the **x** on the tab.

6 In the Instance Detail view, click the **Trace** tab to see the stack trace.

7 Select the `LeakExample.addButtonToPanel()` row, right-click, and select **Show Allocated At Source** to examine the source code for this allocating method.

The Source view displays the *LeakExample.java* source code. The line that allocates the `JButton` is selected. It is located within the `addButtonToPanel()` method. Below this method is the method that is supposed to remove the buttons: `removeButtonFromPanel()`.



Notice that the line of code that removes buttons from the `buttons[]` array is encased in an `if` statement; in essence, it is missing. You have confirmed that the `buttons[]` array is the live object that is holding the loiterers in memory.

Note In Step 5: Running the Memory Leak Session with Improved Code you learn how to remedy this problem, by running the code with the `fix` program argument, thus proving that the original hypothesis is correct.

8 Close the Source view.

9 **Optional**: To find out how much memory is consumed by the loiterer, review the Heap Memory column in the Instances view and the Keep Alive Size column in the Instance Detail view.

Note The **~** sign in the Instances view's Keep Alive Size column indicates an estimated value for this metric. To calculate the actual size, right-click the `JButton` instance and select **Calculate Actual Keep Alive Size** from the list.

You may be surprised at how many instances are held in memory by a single `JButton` instance. The Keep Alive Size for a `JButton` instance is 12,512.

Note These numbers may vary depending on which JVM you used to run the Leak Example.

10 Close the Instances and Instance Detail views.

Step 5: Running the Memory Leak Session with Improved Code

In this step, you need to force the execution of the code that removes the buttons from the `buttons[]` array. The *LeakExample* demo contains the fixed line of code; you just need to add a program argument to your session settings to activate it. Note that in this example, removing the loiterer does not free all the memory, because the memory calculation includes some recursive references and because other objects in the program continue to need some of the objects referenced by the loiterer.

After you rerun the session with the fixed code, you can clearly see in the Instances view that both the Recorded Count and Dead Count for `JButton` are now 0, as predicted by the hypothesis. However, in a real-life scenario where the effect of changes is more widespread, you may need to compare the snapshots to see all the differences. This tutorial walks you through how to do that comparison.

To verify the fixed code:

1. In the JProbe Console, click the **Configurations** tab.
2. Click **Java Application** > **LeakExample**.

   The **LeakExample_Mem_Settings.jpl** tab opens in the View area.
3. Click **Edit**.

   The LeakExample_Mem_Settings.jpl tab is displayed in edit mode, with the Summary tab on the foreground.
4. Click the **Edit** link beside Application Arguments.
5. In the Java Application tab, click the browse button beside the Application Arguments field.
6. In the Application Arguments dialog box, type `fix` in the upper field and click **OK**.

   The argument appears in the Application Arguments field.
7. To save the configuration file, click **Save**.
8. To integrate your application with JProbe, click **Integrate**.
9. In the Integration Wizard click **Save** to save the new startup script, then click **Next**.
10. Run the LeakExample startup script and connect JProbe; click **Run & Connect**.
11. Follow the instructions in Step 2: Running the Memory Leak Session to exercise the same use case.

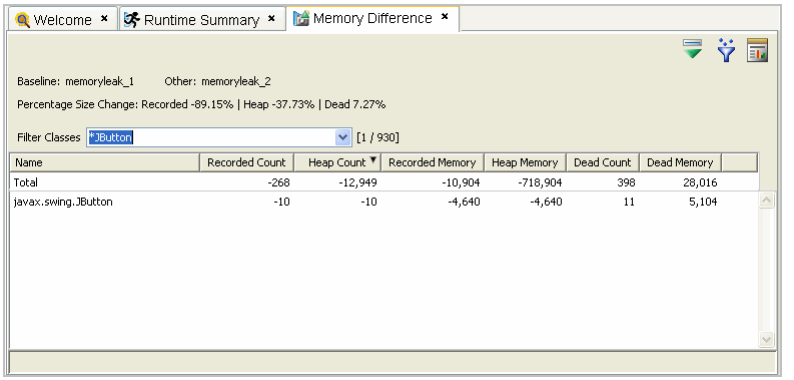
12 In the navigator, right-click the new snapshot with the improved code and select **Snapshot Differencing**.

The Memory Difference dialog box opens. The new snapshot is displayed in the Snapshot to Compare list.

13 Select the original snapshot with the loitering objects from the **Baseline Snapshot** list.

14 Click **OK**.

The Memory Difference view appears, displaying the differences in data between the two snapshots. Use the filters to display only the `JButton` objects.



You can see that in the new snapshot (obtained by running with the `fix` argument) there are ten fewer `JButton` objects than in the original snapshot. The Recorded Count and Heap Count are -10, and the Heap Memory is reduced by approximately 4,640 bytes. Therefore, removing the JButtons from the `JButton` array fixed the problem.

In the upper side of the view, you can also see that the *LeakExample* code now uses memory more efficiently. The improved code reduced the Recorded Memory use by 89.15%.

Network Tutorial

The *Network* program illustrates how over-allocating short-lived objects can cause the garbage collector to run longer than necessary. Garbage collection takes time and resources.

This tutorial demonstrates how to use JProbe to identify excessive garbage collections in your code. The fixed code shows that it is often more efficient to reuse objects or cache data.

Note **The values cited in this tutorial reflect the Network example running on Windows XP with Sun JDK delivered with JProbe.** You may see different values on your system, but the improvement in garbage collection overhead should still be evident.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	*Network.class*
Use Case:	Connect from a client to a server, query a database, and return a result to the client.
Architecture:	See the comments in the *Network.java* source file.
Hypothesis:	The program does not create unnecessary temporary objects.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Network Session
- Step 2: Running the Network Session
- Step 3: Identifying Large Allocations of Short-Lived Objects
- Step 4: Investigating Large Allocations of Short-Lived Objects
- Step 5: Running the Network Session with Improved Code

Step 1: Setting Up the Network Session

To run a garbage collection analysis, you need to set up the session in the JProbe Console. The following procedures mention only the settings that you need to change or verify. If a setting is not mentioned, leave it blank or in its default state. The procedure assumes that you are running JProbe locally on your computer.

To set up the session:

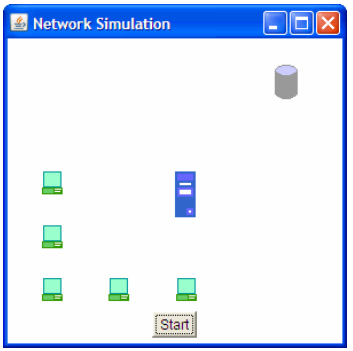
1. In the JProbe Console, click the **Configurations** tab.
2. Expand **Java Application**.

   You can see that JProbe includes a pre-defined configuration file for the Network demo.
3. Click **Demo Network Memory Analysis**.

   The Network_Mem_Settings.jpl tab is displayed in the View area.
4. Click through the configuration tabs to review the settings.

   Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for the Network demo.
5. To run your application startup script and also connect the JProbe Console, click **Run & Connect**.

   The JProbe Execution Console opens, then the *Network Simulation* program starts, displaying a window with icons representing elements of a network. You are now ready to run a Memory analysis session.



Step 2: Running the Network Session

In this step, you exercise the use case on the *Network Simulation* program. All you need to do is to click the **Start** button in the *Network Simulation* program; the program simulates clients (threads) connecting to a server and querying a database. It runs to completion in about one minute (depending on your system setup), generating the data that you need to assess the performance of the garbage collector.

Note This procedure assumes that the *Network Simulation* program is already running (for instructions on how to execute the startup script, see step 5 in section "Step 1: Setting Up the Network Session" on page 29.

Alternatively, you can run the startup script from the command line:

In a Windows command window: `Network.bat`

In a UNIX or Linux shell, from the directory where you saved the startup script: `./Network.sh`

To run the session:

1. In the JProbe Console, on Runtime Summary view select **Five Minutes** from the *Pools and GC Interval* list.
2. Click **Set Recording Level** on the toolbar.
3. In the Set Recording Level dialog box, select **Record Allocations, Stack Traces, and Garbage Data For All Instances**, then click **OK**.

   This enables the recording of allocations, traces and garbage collection on all instances.
4. In the Network Simulation program window, click **Start**.

   When the red lines disappear in the Network Simulation window, the program has finished.
5. Click **Set Recording Level** on the toolbar.
6. In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

   JProbe takes a snapshot and displays it in the Snapshots Navigator.
7. In the Network Simulation window, click **Stop**.
8. Close the Network Simulation window.

JProbe disconnects from the running session. After a few seconds, the Instances view appears, displaying instances that were created during the session.

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

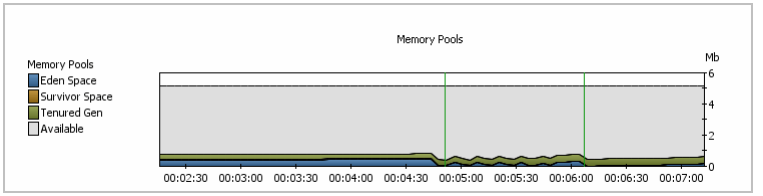
Step 3: Identifying Large Allocations of Short-Lived Objects

The Heap Usage Chart indicates an excessive number of garbage collections.

To identify the short-lived objects:

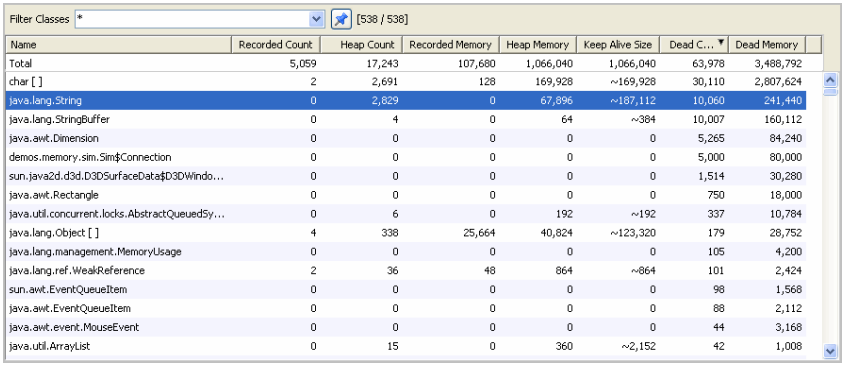
1. In the Runtime Summary view, click the Memory Pools tab.

   The peaks and valleys in the Memory Pools graph show that the objects being collected are not alive very long before they are garbage collected.



2. In the Instances view, select **Dead Count** from the *Investigate by* list.
3. Click the **Dead Count** column header twice to sort the table by the classes with the most garbage collected instances.

   The top classes by Dead Count can vary by JVM and platform, but typically include `char[]`, `String`, `StringBuffer`, and `Sim$Connection`. Few or none of the recorded instances of these classes remain at the end of the session.



Filter Classes * [538 / 538]

Name	Recorded Count	Heap Count	Recorded Memory	Heap Memory	Keep Alive Size	Dead C...	Dead Memory
Total	5,059	17,243	107,680	1,066,040	1,066,040	63,978	3,488,792
char [ ]	2	2,691	128	169,928	~169,928	30,110	2,807,624
java.lang.String	0	2,829	0	67,896	~187,112	10,060	241,440
java.lang.StringBuffer	0	4	0	64	~384	10,007	160,112
java.awt.Dimension	0	0	0	0	0	5,265	84,240
demos.memory.sim.Sim$Connection	0	0	0	0	0	5,000	80,000
sun.java2d.d3d.D3DSurfaceData$D3DWindo...	0	0	0	0	0	1,514	30,280
java.awt.Rectangle	0	0	0	0	0	750	18,000
java.util.concurrent.locks.AbstractQueuedSy...	0	6	0	192	~192	337	10,784
java.lang.Object [ ]	4	338	25,664	40,824	~123,320	179	28,752
java.lang.management.MemoryUsage	0	0	0	0	0	105	4,200
java.lang.ref.WeakReference	2	36	48	864	~864	101	2,424
sun.awt.EventQueueItem	0	0	0	0	0	98	1,568
java.awt.EventQueueItem	0	0	0	0	0	88	2,112
java.awt.event.MouseEvent	0	0	0	0	0	44	3,168
java.util.ArrayList	0	15	0	360	~2,152	42	1,008

We know that `StringBuffer` instances may be created when `String` objects are concatenated, and that both of these use `char[]`. Therefore, `String` and `Sim$Connection` are good candidates for further investigation.

Step 4: Investigating Large Allocations of Short-Lived Objects

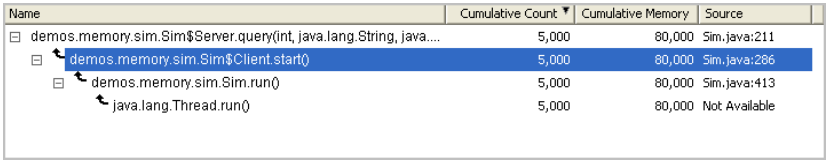
In this step you investigate the short-lived objects identified in Step 3: Identifying Large Allocations of Short-Lived Objects by looking at the source code. Remember that in this example the code contains the fixed code as well. The problem areas are identified in the code comments.

Note To discover where instances are allocated, drill down on an allocation hotspot to display its stack trace in the Merged Allocation Points view. You can then look for your allocating method in the stack trace and drill down on it to see the source code.

To investigate the garbage collected objects:

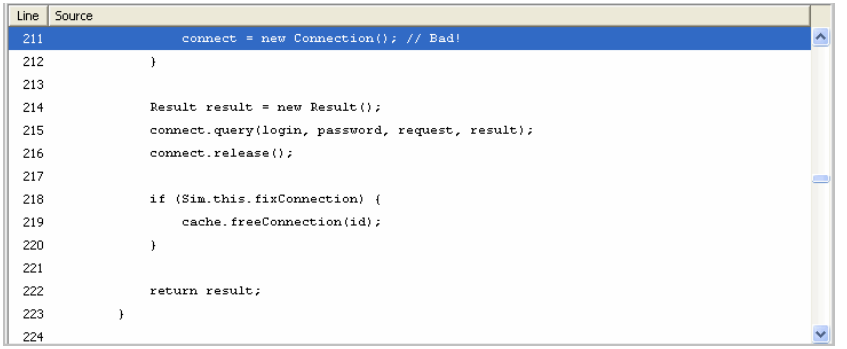
1. Select `Sim$Connection`, right-click, and select **Open Merged Allocation Points View**.
2. Right-click My Code and select **Replace Category with More Detail**.

   The upper pane of the Merged Allocation Points view displays the `Sim$Server.query` method, which allocates instances of `Sim$Connection`. The Source column indicates the line of code where this method occurs.



3 Select `Sim$Server.query`, right-click, and select **Show Allocated At Source**.

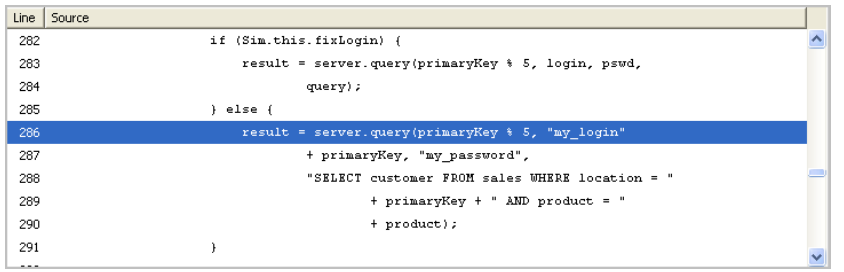
The Source view opens, displaying the *Sim.java* source code at the line indicated in the Source column (211).



Notice that each time a client sends a query to the server, the server creates a new connection to the database that lasts until the connection is terminated. Also notice that `Connection` is an inner class of `Sim`.

4 In the Merged Allocation Points view (lower pane), select `Sim$Client.start`, right-click, and select **Show Allocated At Source**.

The Source view opens, displaying the *Sim.java* source code at the line indicated in the Source column (286).


Line
Source
282 if (Sim.this.fixLogin) {
283 result = server.query(primaryKey % 5, login, pswd,
284 query);
285 } else {
286 result = server.query(primaryKey % 5, "my_login"
287 + primaryKey, "my_password",
288 "SELECT customer FROM sales WHERE location = "
289 + primaryKey + " AND product = "
290 + product);
291 }

Notice that each time a client sends a query to the server, `String` objects are concatenated. This is the source of most of the `String` and `StringBuffer` instances that we saw in the Instances view. You can now proceed with fixing the code.

Step 5: Running the Network Session with Improved Code

Two key problems were discovered in the code: temporary connection objects are created for each connection to the database, and temporary strings are created to pass login, password, and query information. The program contains fixes to reduce the number of temporary objects. You need to add program arguments to use the fixed code. You can run the code with one or both of the fixes.

For the connection issue, use the `-fc` application argument. This problem is solved by implementing a cache. If you review the cache fix in *Sim.java*, you will notice that there are actually three caching options documented: `SimpleCache`, `PoolCache`, and `LocalCache`. By default, the `LocalCache` fix is used. If you change the cache, you need to recompile the program.

For the login issue, use the `-fl` application argument. The problem is solved by introducing static `String` and `StringBuffer` classes that can be reused.

To verify that the fixed code improves memory use:

1. In the Configurations Navigator, right-click **Demo Network Memory Analysis**, and select **Copy settings for Memory Analysis**.

   The settings are copied from the original configuration into *Demo Network Memory Analysis (1)*; the new configuration appears in the Configurations Navigator.

2. Select Demo Network Memory Analysis (1) in the Configurations Navigator.

   The Configuration Details tab is displayed in the View area.

3. Click **Edit**.

4. In the Configuration Name text box, type *NetworkFixed* as the name for the new configuration.

5. Click the **Java Application** tab to change the program argument.

   a. Click the browse button beside the Application Arguments field.

   In the upper field of the Application Arguments dialog box, type:

   ```
   -fl -fc
   ```

b Click **Parse Arguments**.

The arguments appear in separate lines in the lower list.

c Click **OK**.

The Application Arguments field displays the program arguments.

6 Click **Save** then click **Save As** to save the configuration file (*NetworkFixed_Mem_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

7 Click **Integrate**.

The Integration Wizard appears. JProbe validates the configuration file and creates a startup script file (for example, in Windows: *NetworkFixed.bat*, and in UNIX/Linux systems: *NetworkFixed.sh*).

Tip It is a good practice to save the script files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

8 In the Integrate tab, use the browse button to navigate to your working directory, and click **Save** to save the startup file.

9 Click **Next**.

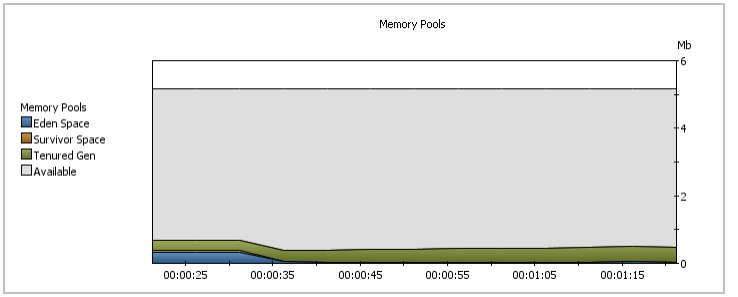
The Run Configuration tab is displayed.

10 Select the **Close syscon window on exit** check box, and click **Run & Connect**.

The JProbe Execution Console opens, then the *Network Simulation* program starts, displaying a window with icons representing elements of a network. You are now ready to run a Memory analysis session using the improved code.

11 Follow the instructions in Step 2: Running the Network Session to exercise the same use case.

In the Memory Pools graph, the peaks and valleys on the graph are less pronounced, which suggests that fewer objects are collected each time the garbage collector runs.



12 In the Instances view, if necessary, select Dead Count from the *Investigate by* list.

13 In the lower pane, click the **Dead Count** column header twice to sort the table by the classes with the most garbage collected instances.

Filter Classes * [535 / 535]

Name	Recorded Count	Heap Count	Recorded Memory	Heap Memory	Keep Alive Size	Dead C...	Dead Memory
Total	5,025	17,252	106,752	1,067,088	1,067,088	8,420	230,048
java.awt.Dimension	0	0	0	0	0	3,367	53,872
sun.java2d.d3d.D3DSurfaceData$D3DWindo...	0	0	0	0	0	1,080	21,600
java.awt.Rectangle	0	0	0	0	0	862	20,688
java.lang.Object [ ]	5	338	25,760	40,864	~123,024	417	32,512
sun.awt.EventQueueItem	0	0	0	0	0	335	5,360
java.util.concurrent.locks.AbstractQueuedSy...	0	5	0	160	~160	329	10,528
java.lang.ref.WeakReference	1	36	24	864	~864	316	7,584
java.awt.EventQueueItem	0	0	0	0	0	310	7,440
java.awt.event.MouseEvent	0	0	0	0	0	133	9,576
java.awt.event.InvocationEvent	0	0	0	0	0	128	7,168
java.util.HashMap$Entry	0	628	0	15,072	~23,544	126	3,024
sun.java2d.pipe.Region	1	3	32	96	~96	116	3,712
java.awt.Point	0	0	0	0	0	116	1,856
char [ ]	2	2,688	128	169,824	~169,824	108	7,104
java.awt.Event	0	0	0	0	0	87	5,568
java.awt.event.PaintEvent	0	0	0	0	0	69	2,208
java.awt.geom.AffineTransform	1	12	64	768	~768	65	4,160
sun.java2d.SunGraphics2D	1	1	200	200	264	65	13,000
java.lang.management.MemoryUsage	0	0	0	0	0	65	2,600
sun.awt.RepaintArea	0	4	0	64	~160	49	784
java.awt.Rectangle [ ]	0	4	0	96	~96	49	1,176
java.lang.String	0	2,826	0	67,824	~186,936	39	936
byte [ ]	2	27	32	27,504	~27,504	32	3,168

The Dead Count of `String` now reports 39 instances, down from more than 10,000 instances in the original example. The total number of Dead instances has also decreased, from 33,000 in original example to 8,420 in the fixed code.

You will notice the following:

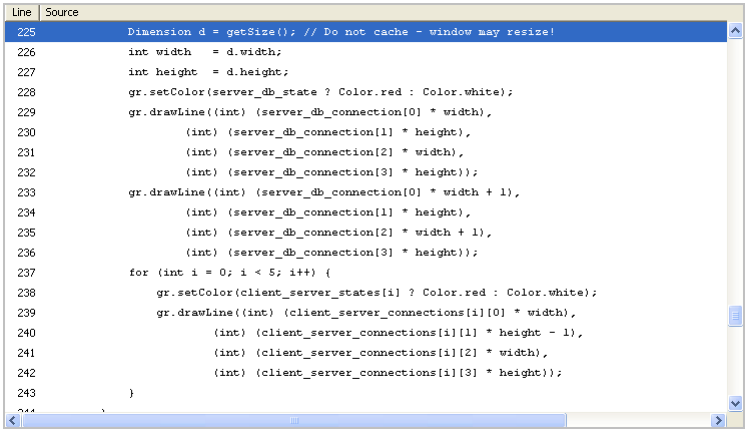
- The `String` count is much lower (39).
- The `Sim$Connection` count is even lower (5).
- `java.awt.Dimension` now has the highest dead count.

14 To investigate `java.awt.Dimension`, select it, right-click, and select **Open Merged Allocation Points View**.

15 In the upper pane of the Merged Allocation Points view, right-click **Sytem Code** end select **Expand To Next Branch Point**.

16 Right-click **My Code** and select **Replace Category With More Detail**.

17 Select the method with the large number of instances (3,345), right-click, and select **Show Allocated At Source**.


Line
Source
225 Dimension d = getSize(); // Do not cache - window may resize!
226 int width = d.width;
227 int height = d.height;
228 gr.setColor(server_db_state ? Color.red : Color.white);
229 gr.drawLine((int) (server_db_connection[0] * width),
230 (int) (server_db_connection[1] * height),
231 (int) (server_db_connection[2] * width),
232 (int) (server_db_connection[3] * height));
233 gr.drawLine((int) (server_db_connection[0] * width + 1),
234 (int) (server_db_connection[1] * height),
235 (int) (server_db_connection[2] * width + 1),
236 (int) (server_db_connection[3] * height));
237 for (int i = 0; i < 5; i++) {
238 gr.setColor(client_server_states[i] ? Color.red : Color.white);
239 gr.drawLine((int) (client_server_connections[i][0] * width),
240 (int) (client_server_connections[i][1] * height - 1),
241 (int) (client_server_connections[i][2] * width),
242 (int) (client_server_connections[i][3] * height));
243 }

You can see that the instances are the result of a `getSize` call on the main *Network Simulation* window. Because it can be resized, the dimensions should not be cached.

2

Performance Analysis Demos

This chapter provides a summary of the Performance demo applications that ship with JProbe, and contains tutorials for some of these applications.

The source code and compiled classes for the Performance demos are located in the *<jprobe_home>/demos/performance* directory.

This chapter contains the following sections:

Summary of Demos for Performance 40
Philosopher's Diner Tutorial 42
Polynomial Tutorial 47

Summary of Demos for Performance

The following table describes the purpose of the example applications.

Java SE Application	Purpose	More Information
Diner.class	The application deadlocks. This example shows how you can use JProbe to identify the threads involved in a deadlock.	See the notes in *Diner.java*. **Tutorial:** "Philosopher's Diner Tutorial" on page 42.
Files.class	Compares the performance of two algorithms: a buffered Reader/Writer versus an unbuffered DataInputStream/ Data OutputStream.	See the notes in *Files.java*.
MethodCalls.class	This application demonstrates the JProbe's ability to track method calls by allowing the user to control the number of calls to specific methods. Each button has a corresponding `ActionListener`. When a button is pressed, the `actionPerformed` method displays a message in the text area. The user can see how Performance tracks method calls by comparing Performance's reported number of calls to each method and the data displayed in the text area.	See the notes in *MethodCalls.java*.
Objects.class	Compares the performance of using objects versus primitives.	See the notes in *Objects.java*.

Java SE Application	Purpose	More Information
Polynomial.class	The application calculates a polynomial expression using one of two algorithms. This example shows how you can use JProbe to compare performance and identify the more efficient algorithm.	See the notes in *Polynomial.java*. **Tutorial**: "Polynomial Tutorial" on page 47.
Strings.class	This application uses two alternative approaches to strip embedded tabs from strings. You can track the two methods and compare their performance.	See the notes in *Strings.java*.

JProbe also ships with a Java EE demo application called JProbe Game Pack Demo for Java EE. For more information, see "JProbe Game Pack for JavaEE" on page 67.

Philosopher's Diner Tutorial

In this tutorial, you investigate a deadlock situation. The symptom of the problem is that the program hangs and does not terminate.

Based on the classic Dining Philosophers deadlock demonstration, the tutorial program simulates five Philosophers seated around a table, each with a bowl of rice in front of him or her. To eat their rice, there are only five chopsticks available to share among the Philosophers, one placed between each pair of Philosophers. To eat the rice, a Philosopher must have two chopsticks. Once a Philosopher is finished using a chopstick, that chopstick is available to the Philosopher beside him or her. The eventual result of this sharing is a deadlock, when each Philosopher waits indefinitely for another chopstick to become available.

This tutorial illustrates how you can detect where a thread causing a deadlock is executing within your own code.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Diner Session
- Step 2: Running the Diner Session
- Step 3: Investigating the Deadlock

Step 1: Setting Up the Diner Session

In this step, you set up a session to detect deadlock situations. You use the JProbe Console to create the session settings. The following procedures mention only the settings that you need to change or verify. If a setting is not mentioned, leave it blank or in its default state. The procedure assumes that you are running JProbe locally on your computer.

To set up the session:

1. In the JProbe Console, click the **Configurations** tab.
2. Expand **Java Application**.

   You can see that JProbe includes a pre-defined configuration file for the Diner demo.
3. Click **Demo Diner Performance Analysis**.

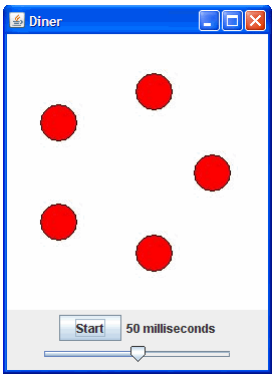
   The Diner_Perf_Settings.jpl tab is displayed in the View area.

4 Click through the configuration tabs to review the settings.

Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for the Diner demo.

5 To run your application startup script and also connect the JProbe Console, click **Run & Connect**.

The JProbe Execution Console opens, then the *Diner* program starts, displaying a window with icons representing the philosophers. The slider controls the number of milliseconds that a Philosopher waits ("sleeps") between chopstick attempts ("eating"). You are now ready to connect a JProbe Console and look for a deadlock.



Step 2: Running the Diner Session

In this step, you run the Diner program until a deadlock occurs. That said, in some cases a deadlock may not occur because the program is sensitive to timing on the computer and in the JVM. If the program does not deadlock, stop the program, adjusting the slider to change the timing, then start the program again.

Note This procedure assumes that the *Diner* program is already running (for instructions on how to execute the startup script, see step 5 in section "Step 1: Setting Up the Diner Session" on page 42.

Alternatively, you can run the startup script from the command line:

In a Windows command window: `Diner.bat`

In a UNIX or Linux shell, from the directory where you saved the startup script: `./Diner.sh`

To run the session:

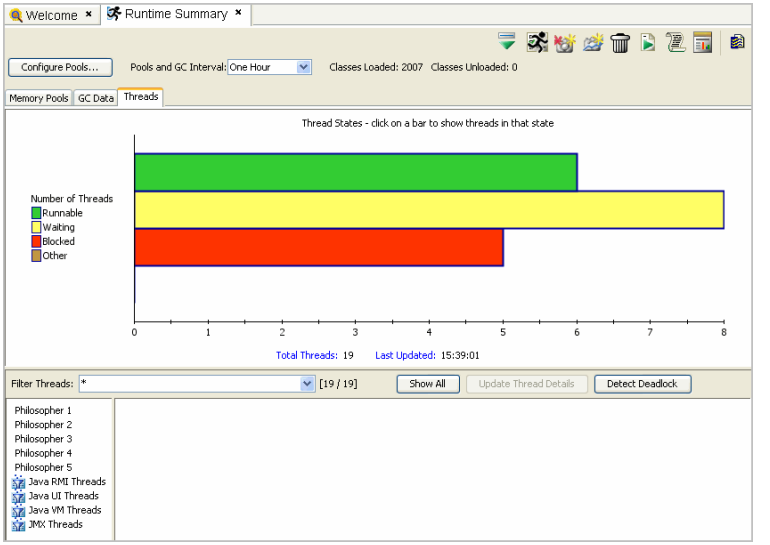
1. In the *Diner* program window, click **Start**, and let the program run until a deadlock occurs.

   The program stops when a deadlock occurs because each philosopher has only one chopstick.

   Tip If a deadlock does not occur at all, click **Stop**, adjust the slider to a lower value (to change the timing), then click **Start** to rerun the program. Repeat until a deadlock occurs.

2. In the JProbe Console, on the Runtime Summary view click the Threads tab.

   The chart in the upper pane shows that at least five threads are blocked. The thread list (in the lower pane) shows the current threads in the program.



3 Click **Detect Deadlock**.

If no deadlock has occurred, the No Deadlock Detected dialog appears, informing you that *JProbe did not detect any deadlocks*.

If a deadlock has occurred, the Deadlock Detected dialog appears, informing you that *JProbe detected a deadlock and has reduced the threads list to the threads involved in the deadlock*. Click **OK**. The thread list updates to show only the deadlocked threads, which in this case are only the Philosopher threads.

Step 3: Investigating the Deadlock

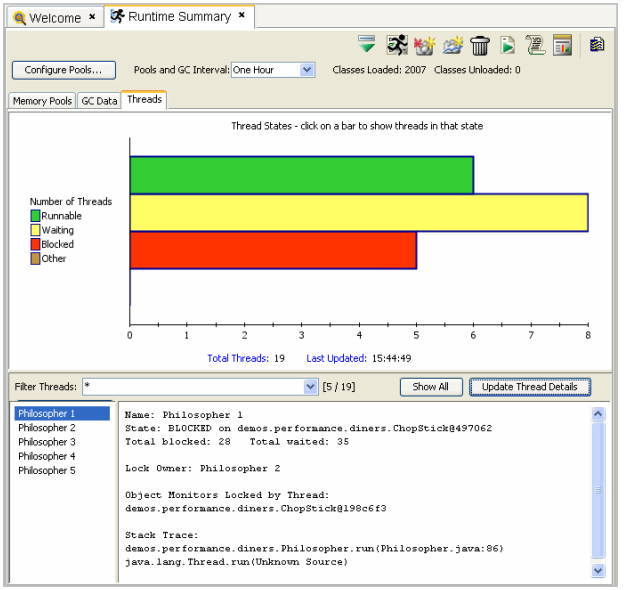
As presented in Step 2: Running the Diner Session, the JProbe Execution Console reported deadlocks among the Philosopher threads.

To see where each thread is deadlocked:

1 In the Runtime Summary view, click the Threads tab (if not already selected).

2 Select a thread from the thread list and click **Update Thread Details**.

The status area is updated with information regarding the selected thread. It shows that the thread is blocked on one of the chopstick objects and that it is currently owned by another Philosopher.



3 Investigate all the deadlocked threads by repeating step 2 for each deadlocked thread in the list.

You can see that Philosopher 1 is waiting for a lock owned by Philosopher 2, who is waiting for a lock owned by Philosopher 3, and so forth. This shows that this cycle is causing the deadlock.

The status area also shows the stack trace for each thread. In this case, each thread is in the `Philosopher.run()` method in `Philosopher.java`.

Tip If line numbers are not shown, you must recompile your program with line information (for example, `javac -g`). If line numbers are still not shown, you may need to disable the JVM's JIT (Just In Time) or optimizing compiler.

Polynomial Tutorial

The *Polynomial* tutorial illustrates how an inefficient algorithm can significantly impact the performance of your code, and how you can use JProbe to compare performance and identify the more efficient algorithm.

Tip You only want to optimize algorithms in the critical path of your program; there is no point tuning algorithms that are called very rarely. You also need to evaluate the overall impact on the runtime of the program. If an inefficient algorithm takes a total of a few seconds to execute, you may make it run faster, but the impact on the overall runtime of the program will be negligible.

The program runs two algorithms for computing the following polynomial expression:

$a_n * x^n + ... + a_2 * x^2 + a_1 * x + a_0$, where n=750

The original algorithm uses nested loops to calculate x^n. The improved algorithm implements Horner's Rule for evaluating polynomial expressions, that is, it factors out powers of x in the form:

$(((...(a_n * x + a_{n-1})...) x + a_2) x + a_1) + a_0$

Note **The values cited in this tutorial reflect the Polynomial running on Windows XP with Sun delivered with JProbe.** You may see different values on your system, but the performance improvement between the algorithms should still be evident.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	*Polynomial.class*
Use Case:	Calculate a polynomial expression.
Architecture:	Both polynomial calculations are in the `evaluate()` method. To run the program, you need to set a program argument: • `N` = Use nested loops (original algorithm) • `H` = Use Horner's Rule (alternate algorithm)
Hypothesis:	Horner's Rule is faster.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Polynomial Session
- Step 2: Running the Polynomial Session
- Step 3: Identifying and Investigating the Performance Bottleneck
- Step 4: Running the Polynomial with Improved Code
- Step 5: Measuring the Performance Improvement

Step 1: Setting Up the Polynomial Session

To collect timing data on the original algorithm, you need to set up the session in the JProbe Console. The following procedures mention only the settings that you need to change or verify. If a setting is not mentioned, leave it blank or in its default state. The procedure assumes that you are running JProbe locally on your computer.

To set up the session:

1. In the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.

2. Click **Java Application > Java Application**.

3. Provide a name for this configuration in the Configuration Name text box (for example, *Polynomial*), then click **Next**.

   The Java Application tab is displayed.

4. In the Main Class area, select the **Execute a class** check box.

5. Click the browse button in the Main Class area and navigate to the *Polynomial.class* file in the *<jprobe_home>/demos/performance/polynomial* directory.

6. Click **OK**.

   The following information is displayed:

   - Main Class: `demos.performance.polynomial.Polynomial`
   - Working Directory: *`<jprobe_home>`*

7. To use the nested loop algorithm, you need to enter a program argument.

   a. Click the browse button beside the Application Arguments field.

   b. In the upper field of the Application Arguments dialog, type: `N`

c Click **Parse Arguments**.

The argument appears in the first line of the lower field.

d Click **OK**.

The Application Arguments field displays the program argument.

8 Click the browse button beside the Classpath field.

9 In the Classpath dialog box, click **Add Working Directory**, then click **OK**.

The working directory appears in the Classpath field.

10 Click **Next**.

The Select a Java Virtual Machine tab is displayed.

11 If you want to change the default JVM, click the browse button beside the Java Executable field, select another JVM in the Java Virtual Machines dialog box, and click **OK**.

Note Ideally, the JVM you choose should be the version that was used to compile your program.

12 Click **Next**.

The My Code tab is displayed.

13 In the Category/Program Name text box type *PolynomialDemo* (which specifies the name of the category in which you want to include your code), then click **Next**.

The Analysis Type tab is displayed.

14 Select the **Performance Analysis** option.

15 On the **General** tab, select the **CPU** timing option.

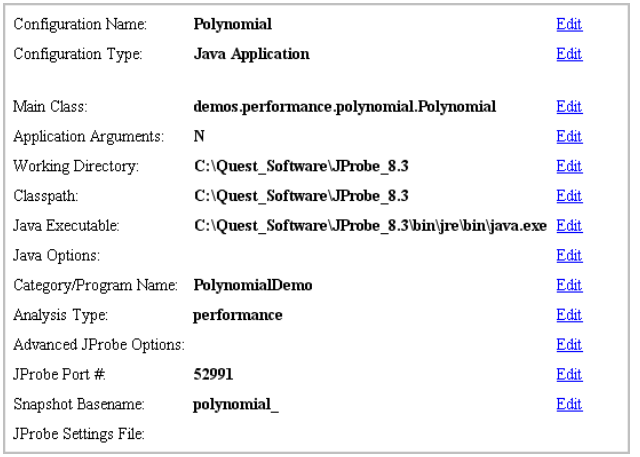
16 On the **Automation** tab, move the slider to select the **Full Encapsulation** option, then click **Next**.

This enables JProbe to collect data for all methods and the methods they call, from the initial JVM start.

The Advanced JProbe Options tab is displayed.

17 In the Snapshot Basename text box type *polynomial_*, and then click **Next**.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.



Configuration Name:	**Polynomial**	Edit
Configuration Type:	**Java Application**	Edit
Main Class:	**demos.performance.polynomial.Polynomial**	Edit
Application Arguments:	**N**	Edit
Working Directory:	**C:\Quest_Software\JProbe_8.3**	Edit
Classpath:	**C:\Quest_Software\JProbe_8.3**	Edit
Java Executable:	**C:\Quest_Software\JProbe_8.3\bin\jre\bin\java.exe**	Edit
Java Options:		Edit
Category/Program Name:	**PolynomialDemo**	Edit
Analysis Type:	**performance**	Edit
Advanced JProbe Options:		Edit
JProbe Port #:	**52991**	Edit
Snapshot Basename:	**polynomial_**	Edit
JProbe Settings File:		

18 To create a JPL file and a startup script, click **Save & Integrate**.

19 Click **Save** to save the configuration file (*Polynomial_Perf_Settings.jpl*) into your working directory.

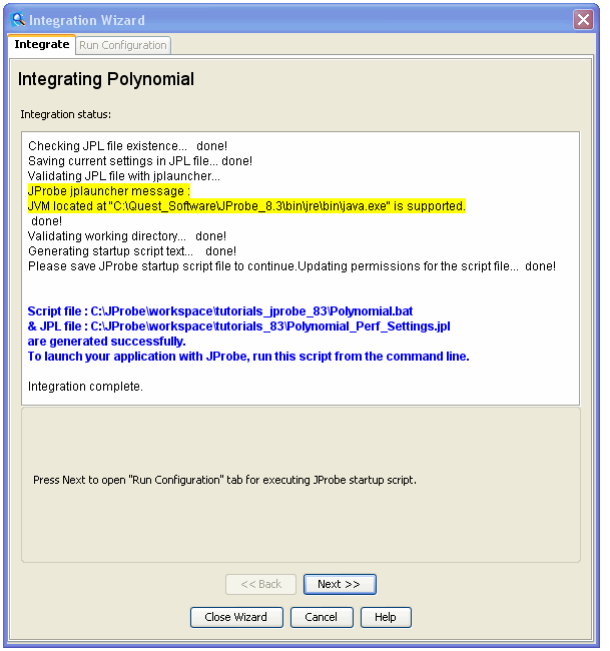
Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

20 In the Integrate tab, use the browse button to navigate to your working directory, and click **Save**.

Tip It is a good practice to save the script files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

JProbe validates the configuration file and creates a startup script file (for example, in Windows: *Polynomial.bat*, and in UNIX/Linux systems: *Polynomial.sh*).


Integration Wizard
Integrate
Run Configuration
Integrating Polynomial
Integration status:
Checking JPL file existence... done!
Saving current settings in JPL file... done!
Validating JPL file with jplauncher...
JProbe jplauncher message :
JVM located at "C:\Quest_Software\JProbe_8.3\bin\jre\bin\java.exe" is supported.
done!
Validating working directory... done!
Generating startup script text... done!
Please save JProbe startup script file to continue.Updating permissions for the script file... done!
Script file : C:\JProbe\workspace\tutorials_jprobe_83\Polynomial.bat
& JPL file : C:\JProbe\workspace\tutorials_83\Polynomial_Perf_Settings.jpl
are generated successfully.
To launch your application with JProbe, run this script from the command line.
Integration complete.
Press Next to open "Run Configuration" tab for executing JProbe startup script.
<< Back
Next >>
Close Wizard
Cancel
Help

21 When the integration is complete, click **Close Wizard**.

You are now ready to run a Performance analysis session.

Step 2: Running the Polynomial Session

In this step, you exercise the use case on the *Polynomial* program. All you need to do is to start the *Polynomial* program. It runs to completion in about one minute (depending on your system setup), generating the data that you need to assess the performance of the “nested loops” algorithm.

To run the session:

1 In the Configurations Navigator, select **Java Application > Polynomial**.

The Polynomial_Perf_Settings.jpl tab appears in the View area.

2 Click **Run & Connect**.

JProbe is searching for currently running engines and JVMs; after a few seconds, the JProbe Console connects to the JProbe Analysis Engine running the polynomial application.

The *Polynomial* program starts and runs in a command window.

When the program is finished, the command window closes. JProbe takes a performance snapshot and displays it in the Snapshots Navigator. The Summary view is also displayed.

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

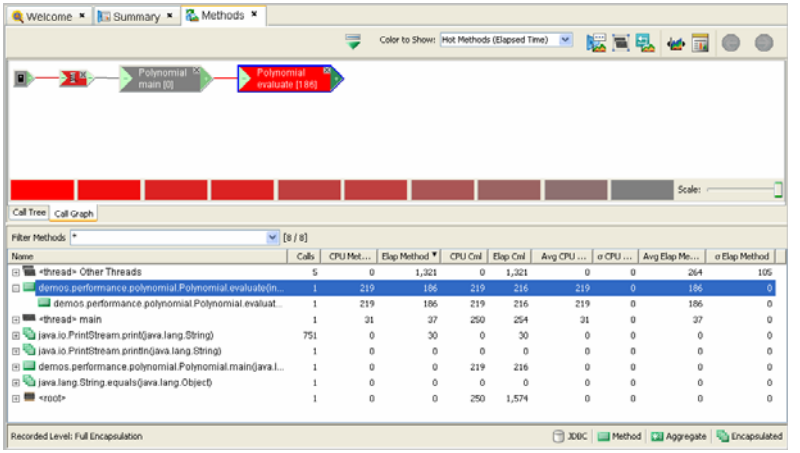
Step 3: Identifying and Investigating the Performance Bottleneck

By default, the snapshot taken at the end of the session is displayed in the Snapshot Navigator. The Summary view presents a list of performance hotspots and guides you on how to investigate any performance hotspots.

To investigate performance bottleneck in your application:

1 In the Summary view, double-click the `evaluate()` method suggested for investigation in the Performance Hotspots section.

The Methods view opens, with the selected `Polynomial.evaluate()` method as the focus method. The method is highlighted in both the graph and the list.



The method was called only once and took 14% of the cumulative time to execute.

Tip If the Methods view shows the actual values, you can modify it to display percentages, by clicking **Tools > Options > Data Display > Performance**, selecting the **Show Percentage Time Values in Methods View Trees** check box, and clicking **OK**.

Note This value may vary when running this tutorial on a different platform. The actual time is less important than the comparative difference between the two algorithms.

2 In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, name the snapshot *Polynomial*, and click **Save**.

3 Close all Performance views.

Step 4: Running the Polynomial with Improved Code

The original algorithm is slower than expected. Based on the hypothesis, the algorithm that implements Horner's Rule should run faster.

To use the Horner's Rule algorithm, you need a new program argument. You can either edit the existing configuration or create a separate configuration to make it easy to switch between tests. In this tutorial, you create a copy of the existing configuration and edit it.

To run and assess the Horner's Rule algorithm:

1 In the Configurations Navigator, right-click **Polynomial**, and select **Copy settings for Performance Analysis**.

The settings are copied from the original configuration into *Polynomial (1)*; the new configuration appears in the Configurations Navigator.

2 Select Polynomial (1) in the Configurations Navigator.

The Configuration Details tab is displayed in the View area.

3 Click **Edit**.

4 In the Configuration Name text box, type *PolynomialFixed* as the name for the new configuration.

5 Click the **Java Application** tab to change the program argument.

a Click the browse button beside the Application Arguments field.

b In the upper field of the Application Arguments dialog box, delete `N` and type: `H`

c Click **Parse Arguments**.

The argument appears in the first line of the lower field.

d Click **OK**.

The Application Arguments field displays the program argument.

6 Click **Save** then click **Save As** to save the configuration file (*PolynomialFixed_Perf_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

7 Click **Integrate**.

The Integration Wizard appears. JProbe validates the configuration file and creates a startup script file (for example, in Windows: *PolynomialFixed.bat*, and in UNIX/Linux systems: *PolynomialFixed.sh*).

Tip It is a good practice to save the script files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

8 In the Integrate tab, use the browse button to navigate to your working directory, and click **Save** to save the startup file.

9 Click **Next**.

The Run Configuration tab is displayed.

10 Select the **Close syscon window on exit** check box, and click **Run & Connect**.

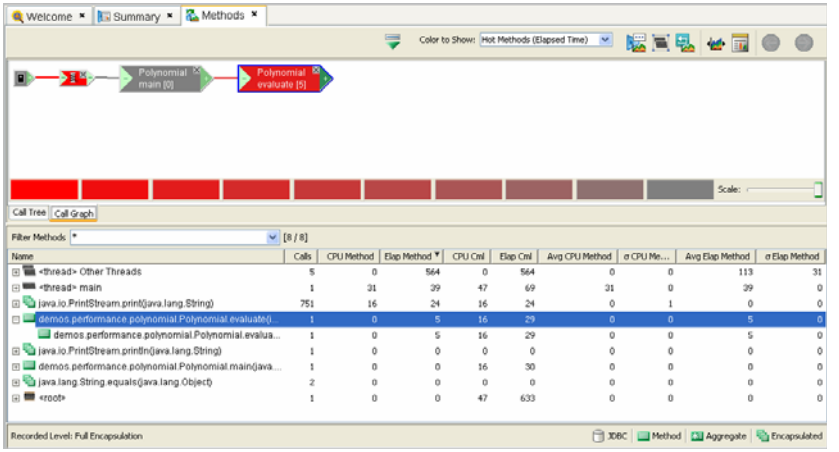
JProbe is searching for currently running engines and JVMs; after a few seconds, the JProbe Console connects to the JProbe Analysis Engine running the polynomial application (using the fixed code).

The *Polynomial* program starts and runs in a command window.

When the program is finished, the command window closes. JProbe takes a performance snapshot and displays it in the Snapshots Navigator. The Summary view is also displayed.

11 In the Summary view, double-click the `Polynomial.evaluate()` method suggested for investigation in the Performance Hotspots section.

The Methods view opens, with the selected `Polynomial.evaluate()` method as the focus method. The method is highlighted in both the graph and the list.



The method was called only once and took 5.6% of the cumulative time to execute.

12 In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, name the snapshot *PolynomialFixed*, and click **Save**.

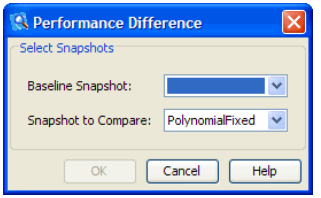
Step 5: Measuring the Performance Improvement

You know that the second algorithm runs much faster than the first one. In this step, you will quantify the performance improvement using the Snapshot Difference window to compare snapshots.

To measure the performance improvement:

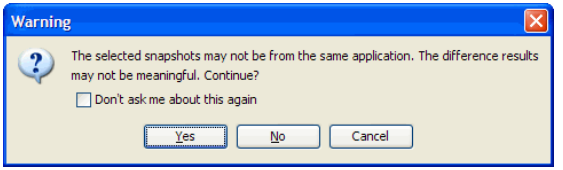
1 In the Snapshots Navigator, right-click the **PolynomialFixed** snapshot and select **Snapshot Differencing**.

The Performance Difference dialog box appears, with PolynomialFixed displayed in the Snapshot to Compare list.

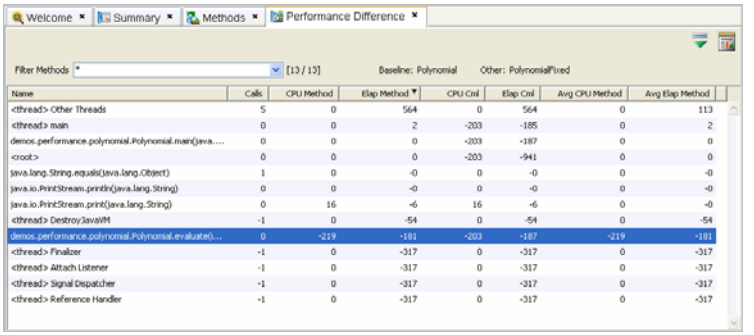


2 Select **Polynomial** from the Baseline Snapshot list and click **OK**.

3 If the following warning dialog appears, click **Yes**.


Warning
The selected snapshots may not be from the same application. The difference results may not be meaningful. Continue?
Don't ask me about this again
Yes
No
Cancel

The Performance Difference view appears, displaying the differences between the two methods.


Welcome
Summary
Methods
Performance Difference
Filter Methods
[13 / 13]
Baseline: Polynomial
Other: PolynomialFixed
Name
Calls
CPU Method
Elap Method
CPU Cml
Elap Cml
Avg CPU Method
Avg Elap Method
<thread> Other Threads 5 0 564 0 564 0 113
<thread> main 0 0 2 -203 -185 0 2
demos.performance.polynomial.Polynomial.main(java.... 0 0 0 -203 -187 0 0
<root> 0 0 0 -203 -941 0 0
java.lang.String.equals(java.lang.Object) 1 0 -0 0 -0 0 -0
java.io.PrintStream.println(java.lang.String) 0 0 -0 0 -0 0 -0
java.io.PrintStream.print(java.lang.String) 0 16 -6 16 -6 0 -0
<thread> DestroyJavaVM -1 0 -54 0 -54 0 -54
demos.performance.polynomial.Polynomial.evaluate()... 0 -219 -181 -203 -187 -219 -181
<thread> Finalizer -1 0 -317 0 -317 0 -317
<thread> Attach Listener -1 0 -317 0 -317 0 -317
<thread> Signal Dispatcher -1 0 -317 0 -317 0 -317
<thread> Reference Handler -1 0 -317 0 -317 0 -317

The number of calls to `Polynomial.evaluate()` did not change, but the Elapsed Cumulative Time has decreased by 181.

Note Negative values represent a performance improvement. Therefore, the Horner's Rule algorithm runs considerably faster than the nested loop algorithm.

3

Coverage Analysis Demos

This chapter provides a summary of the Coverage demo applications that ship with JProbe, and a tutorial for one of these applications.

The source code and compiled classes for the Coverage demos are located in the *<jprobe_home>/demos/coverage* directory.

This chapter contains the following sections:

Summary of Demos for Coverage 58
Adventure Tutorial 59

Summary of Demos for Coverage

The following table describes the purpose of the example applications.

Java SE Application	Purpose	More Information
Adventure.class	A text-based adventure game in which you navigate through a house. The application ships with two test case input files. The test cases do not provide 100% coverage of the program code.	See the notes in *Adventure.java*. **Tutorial:** "Adventure Tutorial" on page 59
SwitchCaseTest.class	This example demonstrates conditional coverage using a simple switch/case statement.	See the notes in *SwitchCaseTest.java*.
TryCatchFinallyTest.class	This example demonstrates how JProbe tracks try-catch-finally blocks.	See the notes in *TryCatchFinallyTest.java*.

JProbe also ships with a Java EE demo application called JProbe Game Pack Demo for Java EE. For more information, see "JProbe Game Pack for JavaEE" on page 67.

Adventure Tutorial

This basic tutorial shows you how to evaluate the effectiveness of two test cases for a text-based adventure game. The test cases are supplied as text files, which are specified in program arguments. The files for the tutorial are available in the *<jprobe_home>/demos/coverage/adventure* directory.

This tutorial does not create a baseline snapshot of the *Adventure* program because the test cases hit methods in all classes. For more information about the baseline coverage snapshot, see the *JProbe User Guide*.

Note **The values cited in this tutorial reflect the Adventure program running on Windows XP with Sun JDK delivered with JProbe.** You may see different values on your system.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	*Adventure.class*
Test Case:	*AdvTest1.txt, AdvTest2.txt*

The tutorial leads you through the following steps:

- Step 1: Setting Your Global Options
- Step 2: Setting Up the Session for the First Test Case
- Step 3: Running the First Test Case
- Step 4: Setting Up and Running the Second Test Case
- Step 5: Merging the Test Case Results
- Step 6: Assessing Your Test Case Coverage

Step 1: Setting Your Global Options

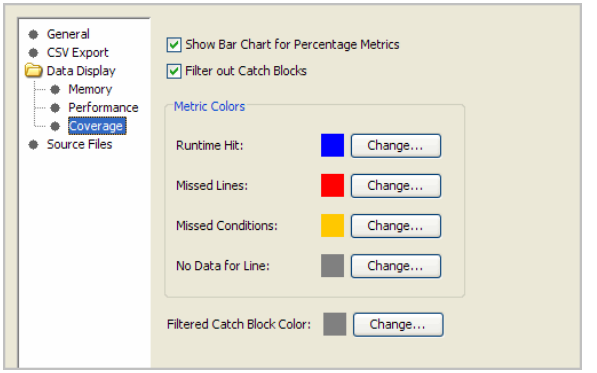
Catch blocks are often hard to test. For this tutorial, we are going to remove the results for catch blocks by setting a global option.

To set global options for the Coverage analysis tool:

1 In the JProbe Console, click **Tools > Options** on the toolbar.

2 Click **Data Display** > **Coverage**.

3 Select the **Filter out Catch Blocks** check box.



4 Click **OK**.

Step 2: Setting Up the Session for the First Test Case

In this step, you create a configuration for the Adventure program using the JProbe Console. The configuration includes the path to a text file that contains the first test case.

To set up the first test case:

1 In the JProbe Console, click the **Configurations** tab.

2 Expand **Java Application**.

You can see that JProbe includes a pre-defined configuration file for the Adventure demo.

3 Click **Demo Adventure Coverage Analysis**.

The Adventure_Cov_Settings.jpl tab is displayed in the View area.

4 Click through the configuration tabs to review the settings.

In the Java Application tab, you can notice that the following program arguments are defined for this test case, in the Application Arguments field: `<jprobe_home>/demos/coverage/adventure/AdvTest1.txt`.

Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for the Adventure demo.

You are now ready to run a Coverage analysis session (test case #1).

Step 3: Running the First Test Case

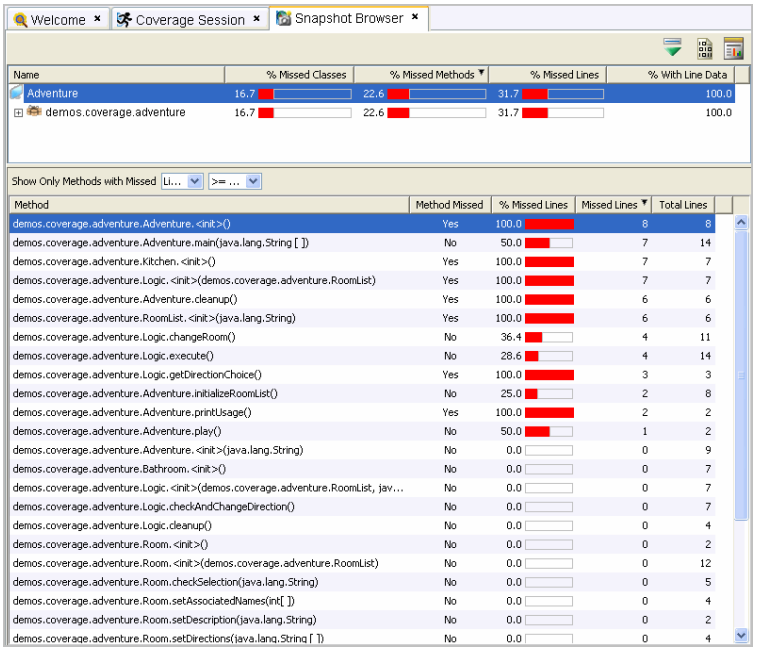
To run the first test case:

- To run your application startup script and also connect the JProbe Console, select **Demo Adventure Coverage Analysis** in the Configurations tab, and click **Run & Connect** in the View area.

  The JProbe Execution Console opens and the program runs using the text file (`AdvTest1.txt`) as test case input for choices.

  When the program is finished, the command window closes. JProbe takes a coverage snapshot and displays it in the Snapshots Navigator and in the Snapshot Browser view.

  Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshots Navigator and select **Open Snapshot**.



Step 4: Setting Up and Running the Second Test Case

To run JProbe with a different test case, you need to change the program arguments for your configuration. Because these are tests that may be modified and re-run, do this by creating a second configuration based on the original, by copying and editing it.

To set up and run the second test case:

1. In the Configurations Navigator, right-click **Demo Adventure Coverage Analysis**, and select **Copy settings for Coverage Analysis**.

   The settings are copied from the original configuration into *Demo Adventure Coverage Analysis (1)*; the new configuration appears in the Configurations Navigator.

2. Select Demo Adventure Coverage Analysis (1) in the Configurations Navigator.

   The Configuration Details tab is displayed in the View area.

3. Click **Edit**.

4. In the Configuration Name text box, type *Adventure_TestCase2* as the name for the new configuration.

5. Click the **Java Application** tab to change the program argument.

   a. Click the browse button beside the Application Arguments field.

   b. In the Application Arguments dialog box, select the existing argument in the lower box and click **Delete**.

   c. Edit the argument in the upper box as follows:

      <*jprobe_home*>`/demos/coverage/adventure/AdvTest2.txt`

      Note If there is a space in your JProbe home directory path, enclose the argument in quotes.

   d. Click **Parse Argument**.

      The argument appears in the first line of the lower field.

   e. Click **OK**.

   The Application Arguments field displays the program argument.

6. Click **Save** then **Save As**, to save the configuration file (*Adventure_TestCase2_Cov_Settings.jpl*) into your working directory.

   Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

7. Click **Integrate**.

The Integration Wizard appears. JProbe validates the configuration file and creates a startup script file (for example, in Windows: *Adventure_TestCase2.bat*, and in UNIX/Linux systems: *Adventure_TestCase2.sh*).

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

8 In the Integrate tab, use the browse button to navigate to your working directory, and click **Save** to save the startup file.

9 Click **Next**.

The Run Configuration tab is displayed.

10 Select the **Close syscon window on exit** check box, and click **Run & Connect**.

The JProbe Execution Console opens and the program runs using the text file as test case input for choices.

When the program is finished, the command window closes. JProbe takes a coverage snapshot and displays it in the Snapshots Navigator and in the Snapshot Browser view.

11 Close the Snapshot Browser view.

Step 5: Merging the Test Case Results

In this step, you merge the snapshots taken during Step 3: Running the First Test Case and Step 4: Setting Up and Running the Second Test Case, to get a complete picture of the coverage provided by these two test cases.

You merge the snapshots using the `jpcovmerge` command line tool. Before you can use this tool, you must save to the disk the two snapshots to be compared.

To merge the snapshots:

1 Save the snapshot taken during Step 3: Running the First Test Case to the disk.

a Right-click the snapshot and select **Save Snapshot As**.

b In the Save As dialog box, save the snapshot as <*working_directory*>*first.jpc*.

2 Repeat step a to step b to save the snapshot taken during Step 4: Setting Up and Running the Second Test Case as <*working_directory*>*second.jpc*.

3 Click **Start > Run** and type the following, then click **OK**:

- In a Windows command window:
  `<jprobe_home>\bin\jpcovmerge <working_directory>\first.jpc`

```
<working_directory>\second.jpc
<working_directory>\merged.jpc
```

- In a UNIX or Linux shell:

```
<jprobe_home>/bin/jpcovmerge <working_directory>/first.jpc
<working_directory>/second.jpc <working_directory>/
merged.jpc
```

Important This command fails if `<jprobe_home>` or `<working_directory>` has spaces in it. You must quote the application/arguments in this case.

4. On the JProbe Console, click **File > Open Snapshot**.
5. In the Open dialog box, select *<working_directory>\merged.jpc* and click **Open**.

   The merged snapshot is selected and its content is displayed in the Snapshot Browser view.

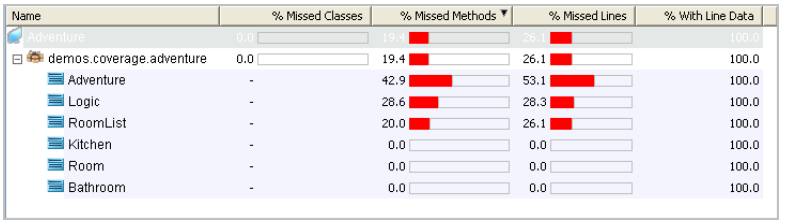
Step 6: Assessing Your Test Case Coverage

After merging your test cases, you are now ready to investigate your results. By default, JProbe displays results in terms of misses. A miss means that the code was not called during your test cases.

To investigate your results:

1. In the Snapshot Browser view, expand the class tree in the upper pane to show all of the classes in `demos.coverage.adventure`.
2. Click the **% Missed Methods** column heading in the upper pane to sort the table from highest-to-lowest percentage of missed methods.

   Methods are missed in three classes: `Adventure`, `Logic`, and `RoomList`. We will investigate the first two classes.

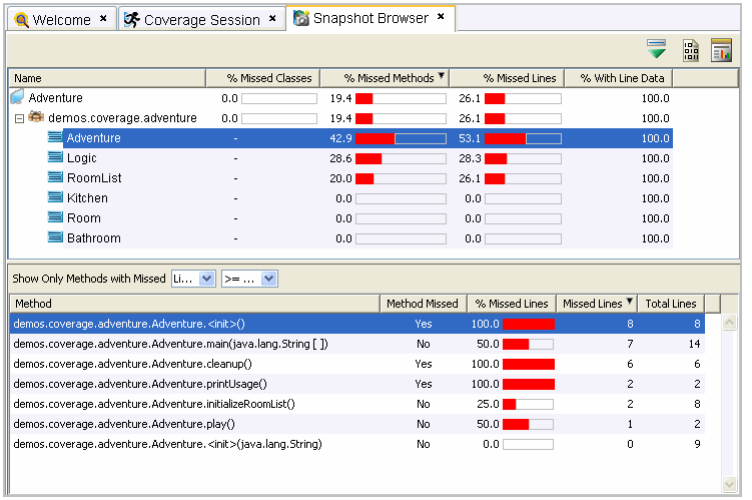


Name	% Missed Classes	% Missed Methods	% Missed Lines	% With Line Data
Adventure	0.0	19.4	26.1	100.0
demos.coverage.adventure	0.0	19.4	26.1	100.0
Adventure	-	42.9	53.1	100.0
Logic	-	28.6	28.3	100.0
RoomList	-	20.0	26.1	100.0
Kitchen	-	0.0	0.0	100.0
Room	-	0.0	0.0	100.0
Bathroom	-	0.0	0.0	100.0

   By default, the Adventure program is selected in the top pane, which means that the lower pane contains all the methods in the program.

3. In the upper pane, select the **Adventure** class.

The methods for the Adventure class are displayed in the lower pane.



4 Select the `main()` method, right-click, and select **Show Source** to open the *<jprobe_home>\demos\coverage\adventure\Adventure.java* source code.

The source code opens at a block of missed lines that are in an `if` statement, so you know that the test case missed a condition. You have identified an opportunity to expand the test suite with a new test case. In this case, you would pass two input files as program arguments.

5 Click the **Down** button at the top of the coverage bar (to the right of the scroll bar) to move to the next set of missed lines.

We can see that code is missed because input is not coming from the console. You can ignore these misses.

6 Continue in this way until you have examined all the missed blocks of code, then move on to the next missed class.

You have identified an area for improvement. If you like, you can modify the test suite and redo the tutorial to see the improvement in overall coverage.

4

JProbe Game Pack for JavaEE

This section describes how to deploy the JProbe Game Pack demo software and presents how to use JProbe with JavaEE applications that require a third-party application server. The Game Pack tutorials describe how to find two different kinds of memory leaks with the Memory analysis tool, how to investigate a performance bottleneck with the Performance analysis tool, and how to assess code coverage with the Coverage analysis tool.

Note Results obtained while running the Game Pack tutorials vary depending on the application server you are using.

This chapter contains the following sections:

Getting Started with the Game Pack Demo 68
Quick Game Pack Evaluation 70
Advanced Game Pack Evaluation 95
Creating a User Account for Game Pack 134
Game Pack Known Issues 135

Getting Started with the Game Pack Demo

Refer to these sections for information about the following topics:

- "Installing the Game Pack Demo" on page 68
- "System Requirements" on page 69
- "Setting Environment Variables" on page 69

Installing the Game Pack Demo

For detailed installation instructions, see the *JProbe Installation Guide*.

If you chose to install examples during the JProbe installation, the Game Pack demo files are installed automatically in the following directory structure:

JPROBE_HOME		
	demos\	
		gamepack\
			build\	Build and class files.
			geronimo-tomcat\	Geronimo-Tomcat 2.1 application server pre-configured with the JProbe Game Pack application.
			lib\	This directory is initially empty. You can place here any Java EE library API (JAR files you need in order to compile the Game Pack code).
			src\	Source code.
			support\	Support files. Includes a *deployment* subdirectory which contains EAR files for deploying Game Pack on all supported application servers.

System Requirements

The Game Pack demo requires the following environment:

- JProbe v.8.3
- One of the following operating systems:
  - Microsoft® Windows® 2003 or XP PRO SP2 or later
  - Red Hat® AS 4.0 or 5.x
  - Sun Solaris® SPARC 9.0 or 10.0
  - IBM AIX® 5L 5.3 or 6.1
- One of the following application servers:
  - Apache Geronimo 2.1.4 with Jetty 6
  - Apache Geronimo 2.1.4 with Tomcat 6
  - JBoss™ 5.1.0
  - BEA® WebLogic® Server 10.3
  - BEA® WebLogic® Server 11g
- Java EE SDK 1.6.0 or later
- One of the following browsers:
  - FireFox
  - Microsoft IE
  - Apple Safari
- Jakarta Ant 1.6.3 or later

For a list of JProbe supported platforms and installation instructions, see the *JProbe Installation Guide*. The guide is available as a PDF file on the DVD and in the JProbe installation directory *<jprobe_install>\doc*.

You can download Ant 1.6.3 or later from the Apache Jakarta Project at: *http://jakarta.apache.org/ant/*. Extract the files to a directory.

Setting Environment Variables

Before you begin, set up the following environment variables:

- `ANT_HOME=<ant_install_dir>`
- `JPROBE_HOME=<jprobe_install_dir>`

- If you are using Geronimo, `GERONIMO_HOME=<geronimo_install_dir>`
- If you are using JBoss, `JBOSS_HOME=<jboss_install_dir>`
- If you are using WebLogic Server, `WL_HOME=<wlserver##_install_dir>`

Add the following paths to your `PATH` environment variable (use the syntax appropriate for your operating system):

- `%ANT_HOME%\bin`
- `%JPROBE_HOME%`
- If you are using Geronimo, `%GERONIMO_HOME%`
- If you are using JBoss, `%JBOSS_HOME%`
- If you are using WebLogic Server, `%WL_HOME%`

Quick Game Pack Evaluation

In order to make easier the evaluation of the Game Pack demo, JProbe includes a Geronimo-Tomcat 2.1 application server pre-configured and integrated with the Game Pack application.

Note If you want to run the Game Pack demo with an application server other than Geronimo-Tomcat 2.1, you need to deploy Game Pack on that application server before running a memory, performance, or coverage analysis. For details, see "Advanced Game Pack Evaluation" on page 95.

The JProbe installer sets up the Geronimo application server and Game Pack in the *<JPROBE_HOME>/demos/gamepack* directory, installs three ready-to-use JPL settings files, and creates three Geronimo-Game Pack configurations, one for each analysis type. These pre-defined configurations are available in the Configurations Navigator, and you just need to run them in order to start a memory, performance, or coverage analysis.

For details, see the following topics:

- "Game Pack Memory Analysis Tutorials" on page 71
- "Game Pack Performance Analysis Tutorial" on page 83
- "Game Pack Coverage Analysis Tutorial" on page 89

Game Pack Memory Analysis Tutorials

The Game Pack Memory tutorials describe how to find two different kinds of memory leaks using the JProbe Memory analysis tool and the pre-defined *Demo Gamepack Memory Analysis* configuration:

- "Loitering Objects Tutorial" on page 71
- "Object Cycling Tutorial" on page 77

Loitering Objects Tutorial

Minesweeper and the Match Game both offer a Loitering Objects mode. The Loitering Objects mode demonstrates how an obsolete container reference can keep session beans in the Java heap long after their usefulness is gone. In this tutorial, you see how you can reclaim memory by removing loitering `GameHandler` objects.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack demo: Match Game
Use Case:	Play a few games without quitting.
Architecture:	When the Start button is selected, create a `GameHandler` object to run the game.
Hypothesis:	The `GameHandler` object is removed from the heap when the game ends.

The tutorial walks you through the following steps:

- Step 1: Starting the Session and the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Shutting Down the Server
- Step 5: Identifying Loitering Objects
- Step 6: Investigating Loitering Objects
- Step 7: Running the Session with Improved Code

Step 1: Starting the Session and the Game Pack

In this step, you start the Geronimo-Tomcat server using the pre-defined startup script shipped with JProbe, then connect JProbe to the Game Pack application running on this server.

To start the session from command line:

1. Type in the following command, depending on your platform:

- In Windows:

  ```
  <JPROBE_HOME>\demos\settings\startWith_MemAnalysis.bat
  ```

- In a UNIX or Linux shell:

  ```
  <JPROBE_HOME>\demos\settings\startWith_MemAnalysis.sh
  ```

  The Geronimo-Tomcat application server and the Game Pack application start in a command line window, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

  Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

2. Open the JProbe Console and click **Attach to Session**.
3. In the Attach to Session dialog box, select the JProbe Analysis Engine and click **OK**.

   JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

To start the session from the JProbe Console:

1. Click the **Configurations** tab and expand **Apache**.

   You can see that JProbe includes a pre-defined configuration file for the Game Pack memory demos.

2. Click **Demo Gamepack Memory Analysis**.

   The Gamepack_Mem_Settings.jpl tab is displayed in the View area.

3. Click through the configuration tabs to review the settings.

   Note These pre-defined settings cannot be modified or deleted. You do not need to integrate your application with JProbe, because a startup script is already provided for this demo.

4. To run the Game Pack startup script and also connect the JProbe Console, click **Run & Connect**.

The JProbe Execution Console opens, the Geronimo-Tomcat application server and the Game Pack application start, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1. In the *Welcome to the JProbe Game Pack Demo Application* page, click **Start using Game Pack**.

   The Game Pack demo login page appears.

2. Enter your **User ID** and **Password**, then click **Login**.

   Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

   You are now ready to run a Memory analysis session.

Step 3: Running the Session

In this step, you work through a use case by playing three consecutive games. It does not matter for the analysis whether you win or lose the games. However, it is important that you start a session before you begin to play, or JProbe will not perform a garbage collection. After you have played three games, you end the session, and JProbe takes a snapshot.

To run a game with the Loitering Objects fault:

1. In the Game Pack demo, click **Play** beside either Minesweeper or Match Game.

   Note Both games have the same loitering object problem.

2. Select the **Loitering Objects** option.

   Note Clicking the link displays the option's definition.

3. In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.

4. In the Set Recording Level dialog box, select **Record Allocations and Stack Traces For All Instances**, then click **OK**.
5. In the Game Pack demo, click **Start** and play the game.
6. When the game ends, play the game twice more (without quitting) for a total of three complete games.
7. Click **Quit**.
8. In the Runtime Summary view, click **Set Recording Level** on the toolbar.
9. In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

   JProbe takes a snapshot and displays it in the Snapshots Navigator.
10. In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.
11. Name the snapshot *loitering_objects* and click **Save**.

    The snapshot name is displayed in the Snapshots Navigator.
12. Click **Detach From Session** .

    The Instances view appears after a few seconds.

    Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.
13. On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Shutting Down the Server

In this step, you close the Geronimo-Tomcat application server.

To stop the Geronimo-Tomcat server:

1. Open a command window and navigate to:

   ```
   <JPROBE_HOME>\demos\settings
   ```
2. Type in the following command:
   - In Windows:

     ```
     startWith_MemAnalysis.bat stop
     ```
   - In a UNIX or Linux shell:

     ```
     startWith_MemAnalysis.sh stop
     ```

3. Type in Username: `system`.
4. Type in Password: `manager`.

   The Geronimo-Tomcat application server shuts down and the JProbe Execution Console closes.

Step 5: Identifying Loitering Objects

In this step, you look for loitering objects in the heap. The Heap Count column is the first place to look. In general, you should expect objects created during a session to be removed at the end of it. In fact, the `GameHandler` objects are not removed, and three instances of this object continue to loiter in the heap.

To identify loitering objects:

1. If the *loitering_objects* snapshot is not open, right-click it in the snapshot navigator and select **Open Instances View**.

   The Instances view appears.
2. In the Filter Classes field, type `*.GameHandler` and press **Enter** to display only the `GameHandler` class.

   The Heap Count for GameHandler is 3, not 0 (zero) as hypothesized.

Step 6: Investigating Loitering Objects

In this step, you find the live object that continues to hold a reference to the loitering instances of `GameHandler` in the heap. You start in the Instances view, then drill down to Instance Detail and the Memory Leak views, and discover that the loitering `GameHandler` objects are being held by an obsolete container reference. To look for a solution, you open the Source view and review the code for the allocating method.

To investigate loitering objects:

1. If the snapshot is not open, right-click it in the snapshot navigator and select the **Open Instances View**.

   The Instances view appears.
2. Select **Heap Count** from the *Investigate by* list.
3. Filter the method list by typing `*.GameHandler` in the Filter Classes field and pressing **Enter**.
4. Select the `GameHandler` class and click **Instance Detail** .

The Instance Detail view opens. The three loitering `GameHandler` objects are displayed in the instances list. You can see the stack trace of method calls in the **Allocated At** column.

5 Click the Trace tab and scroll down the method list to the methods belonging to the subpackages of the `demos.gamepack.web.game` package and find a method called `EJBControllerImpl.startGame()`.

The `startGame()` method calls the `getGameHandlerRemote()` method, which in turn sets off a series of calls to JBoss methods that eventually causes the bean to be created. Therefore, the `getGameHandlerRemote()` method is the most likely candidate for investigation, because it is the last Game Pack method before the series of application server calls.

6 Select the `demos.gamepack.web.game.EJBControllerImpl.getGameHandlerRemote()` method, right-click, and select **Show Allocated At Source**.

7 If you are prompted for the source code, navigate to *<jprobe_home>/demos/gamepack/src/demos/gamepack/web/game*, select *EJBControllerImpl.java*, and click **Open**.

The Source view opens, highlighting line 174 of the code. This line represents the method call from the `getGameHandlerRemote()` method to the application server's `create` method, which creates the beans.

Now that you found the method that creates the loitering `GameHandler` objects, you are close to finding out where the objects should be removed. Scrolling down to line 185, you find that the `resetGameHandlerRemote()` method is the problem method. When the Loitering Objects mode is selected, the `_gameHandlerRemote.removeObject()` method is not called, so the references to the GameHandler objects are never removed.

Tip It is good programming practice to pair your calls to create and remove objects close together.

Step 7: Running the Session with Improved Code

You can re-run Minesweeper or Match Game in `Normal` mode with the corrected code. Repeat Step 1: Starting the Session and the Game Pack through Step 4: Shutting Down the Server, selecting **Normal** mode instead of **Loitering Objects** and not re-naming the snapshot. You will see in the Instances view that the Heap Count for GameHandler is now 0, as predicted by the hypothesis.

This example demonstrates how to use JProbe to identify and remove loitering objects from your code.

Object Cycling Tutorial

The Match Game offers an Object Cycling mode. The Object Cycling mode demonstrates how over-allocating short-lived objects can cause the garbage collector to run more frequently than necessary. Garbage collection takes time and resources. The Normal mode shows that it is often more efficient to reuse strings.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo: Match Game
Use Case:	Play three games without quitting.
Architecture:	**Object Cycling**: Use string concatenation. **Normal**: Append to an existing string buffer object.
Hypothesis:	The program does not create unnecessary temporary string objects.

The tutorial walks you through the following steps:

- Step 1: Starting the Session and the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Shutting Down the Server
- Step 5: Identifying Object Cycling
- Step 6: Investigating Object Cycling
- Step 7: Running the Session with Improved Code

Step 1: Starting the Session and the Game Pack

In this step, you start the Geronimo-Tomcat server using the pre-defined startup script shipped with JProbe, then connect JProbe to the Game Pack application running on this server.

To start the session from command line:

1. Type in the following command, depending on your platform:

- In Windows:

```
<JPROBE_HOME>\demos\settings\startWith_MemAnalysis.bat
```

- In a UNIX or Linux shell:

```
<JPROBE_HOME>\demos\settings\startWith_MemAnalysis.sh
```

The Geronimo-Tomcat application server and the Game Pack application start in a command line window, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

2. Open the JProbe Console and click **Attach to Session**.
3. In the Attach to Session dialog box, select the JProbe Analysis Engine and click **OK**.

   JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

To start the session from the JProbe Console:

1. Click the Configurations tab and expand **Apache**.

   You can see that JProbe includes a pre-defined configuration file for the Game Pack memory demos.
2. Click **Demo Gamepack Memory Analysis**.

   The Gamepack_Mem_Settings.jpl tab is displayed in the View area.
3. Click through the configuration tabs to review the settings.

   Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for this demo.
4. To run the Game Pack startup script and also connect the JProbe Console, click **Run & Connect**.

The JProbe Execution Console opens, the Geronimo-Tomcat application server and the Game Pack application start, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1. In the *Welcome to the JProbe Game Pack Demo Application* page, click **Start using Game Pack**.

   The Game Pack demo login page appears.

2. Enter your **User ID** and **Password**, then click **Login**.

   Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

   You are now ready to run a Memory analysis session.

Step 3: Running the Session

In this step, you turn on garbage monitoring so that you can see how many objects are garbage collected during your use case. You work through a use case by playing three consecutive games. It does not matter for the analysis whether you win or lose the games. However, it is important that you start a use case before you begin to play, or JProbe will not perform a garbage collection. After you have played three games, you end the use case, and JProbe takes a snapshot.

To run a game with the Object Cycling fault:

1. In the Game Pack demo, select **Play** beside Match Game.
2. Select the **Object Cycling** option.

   Note Clicking the link displays the option's definition.

3. In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.

4. In the Set Recording Level dialog box, select **Record Allocations, Stack Traces, and Garbage Data For All Instances** and click **OK**.
5. In the Game Pack demo, click **Start** and play the game.
6. When the game ends, play the game twice more (without quitting) for a total of three complete games.
7. Click **Quit**.
8. In the Runtime Summary view, click **Set Recording Level** on the toolbar.
9. In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

   JProbe takes a snapshot and displays it in the Snapshots Navigator.
10. In the Snapshots Navigator, right-click this snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.
11. Name the snapshot *object_cycling* and click **Save**.

    The snapshot name is displayed in the Snapshots Navigator.
12. Click **Detach From Session** .

    The Instances view appears after a few seconds.

    Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.
13. On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Shutting Down the Server

In this step, you close the Geronimo-Tomcat application server.

To stop the Geronimo-Tomcat server:

1. Open a command window and navigate to:

   ```
   <JPROBE_HOME>\demos\settings
   ```
2. Type in the following command:
   - In Windows:

     ```
     startWith_MemAnalysis.bat stop
     ```
   - In a UNIX or Linux shell:

     ```
     startWith_MemAnalysis.sh stop
     ```

3. Type in Username: `system`.
4. Type in Password: `manager`.

   The Geronimo-Tomcat application server shuts down and the JProbe Execution Console closes.

Step 5: Identifying Object Cycling

In this step, you look for classes and methods that allocate short-lived objects.

The Garbage Collections chart in the GC Data tab displays steep spikes, which means that some set of objects is garbage collected soon after being created. In the Instances view, look for classes with high Dead Count values and no or very few instances still alive. In this example, you can see that many instances of `StringBuffer` were allocated and garbage collected. None of the instances are still alive. When you review the results, you see that most of the `StringBuffer` objects were allocated by `*_StringConcatenation` methods in the `MatchGameRenderer` class.

To identify short-lived objects:

1. If the *object_cycling* snapshot is not open, right-click it in the Snapshots Navigator and select **Open Instances View**.

   The Instances view appears.
2. Select **Dead Count** from the *Investigate by* list, and sort the table by **Dead Count**.

   You can see that many `String` and `StringBuffer` instances are immediately garbage collected (that is, high **Dead Count** values and low **Recorded Count** values).
3. Drill into the Merged Allocation Points and Call Traces views, by selecting the `StringBuffer` instance, right-clicking, and selecting the **Open Merged Allocation Points View** and **Open Call Traces View**, respectively.
4. In the Merged Allocation Points view, right-click My Code and select **Replace Category with More Detail**, then sort by Cumulative Count.

   You can see that most of the `StringBuffer` objects were allocated by `renderGameMap_StringConcatenation`, `renderGamePlay_StringConcatenation`, and `renderSnapshot_StringConcatenation` methods in the `MatchGameRenderer` class, `demos.gamepack.matchgame` package.

Step 6: Investigating Object Cycling

In Java, the JVM converts string concatenations into `StringBuffer` objects, which means that each concatenation creates a new object with a very short life span. This is a less efficient way to handle strings.

Take a look at the source code to see exactly how the application works. The `_StringConcatenation` methods are in the `MatchGameRenderer.java` source code. You can use any editor to examine the allocating methods, but it makes it easier to find the method if your editor has a search feature. Remember that in this case the code contains the fixed code as well.

To investigate the garbage collected objects:

1. Navigate to *<jprobe_home>/demos/gamepack/src/demos/gamepack/matchgame* and open the *MatchGameRenderer.java* file in a source code editor.
2. Search for the top method: `renderGamePlay_StringConcatenation`.
3. Observe that the method contains many string concatenations. If you scroll down to find `renderSnapshot_StringConcatenation` and then `renderGameMap_StringConcatenation`, you can see that these methods also use string concatenation.

Step 7: Running the Session with Improved Code

The file *MatchGameRenderer.java* also contains methods that offer a better way to handle the strings. While you have the source code open, you can scroll to find these improved methods:

- `renderGamePlay_StringBufferAppend`
- `renderSnapshot_StringBufferAppend`
- `renderGameMap_StringBufferAppend`

If you want, you can re-run this tutorial using **Normal** mode, which uses the improved methods. When you check the Instances view, you should see that the number of short-lived `StringBuffer` objects is reduced significantly.

This example demonstrates how to use JProbe to identify and remove object cycling problems in your code.

Game Pack Performance Analysis Tutorial

The Game Pack Performance tutorial (Performance Bottleneck Tutorial) describes how to investigate a performance bottleneck using the JProbe Performance analysis tool and the pre-defined *Demo Gamepack Performance Analysis* configuration.

Performance Bottleneck Tutorial

The Minesweeper game can be played in the Method Time mode. This mode demonstrates a performance bottleneck caused by an inappropriate algorithm that is used for rendering the Minesweeper game board as one large image. For comparison, you can play the game in the Normal mode, which creates the game board as a table containing HTML links to images.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo, Minesweeper Game Entry point of interest: `GameController.doGet`
Use Case:	Run minesweeper and select one tile.
Architecture:	**Method Time**: The game board is constructed with images on the server-side, encoded as a single image, and sent to the browser. You may notice the image jumps when a tile is selected. **Normal**: The game board is created as a table of links to images.
Hypothesis:	Encoding is slow. Creating the game board with links to images will be faster and smoother, especially if the images are cached by the browser.

The tutorial walks you through the following steps:

- Step 1: Starting the Session and the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Shutting Down the Server
- Step 5: Identifying the Performance Bottleneck
- Step 6: Running the Session with Improved Code

- Step 7: Measuring the Performance Improvement

Step 1: Starting the Session and the Game Pack

In this step, you start the Geronimo-Tomcat server using the pre-defined startup script shipped with JProbe, then connect JProbe to the Game Pack application running on this server.

To start the session from command line:

1. Type in the following command, depending on your platform:

- In Windows:

  ```
  <JPROBE_HOME>\demos\settings\startWith_PerfAnalysis.bat
  ```

- In a UNIX or Linux shell:

  ```
  <JPROBE_HOME>\demos\settings\startWith_PerfAnalysis.sh
  ```

  The Geronimo-Tomcat application server and the Game Pack application start in a command line window, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

  Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

2. Open the JProbe Console and click **Attach to Session**.
3. In the Attach to Session dialog box, select the JProbe Analysis Engine and click **OK**.

   JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

To start the session from the JProbe Console:

1. Click the **Configurations** tab and expand **Apache**.

   You can see that JProbe includes a pre-defined configuration file for the Game Pack performance demo.

2. Click **Demo Gamepack Performance Analysis**.

   The Gamepack_Perf_Settings.jpl tab is displayed in the View area.

3. Click through the configuration tabs to review the settings.

   Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for this demo.

4 To run the Game Pack startup script and also connect the JProbe Console, click **Run & Connect**.

The JProbe Execution Console opens, the Geronimo-Tomcat application server and the Game Pack application start, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1 In the *Welcome to the JProbe Game Pack Demo Application* page, click **Start using Game Pack**.

The Game Pack demo login page appears.

2 Enter your **User ID** and **Password**, then click **Login**.

Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

You are now ready to run a Performance analysis session.

Step 3: Running the Session

In this step, you exercise the use case by selecting one tile. It does not matter for the analysis whether the tile reveals a number or a mine. In Method Time mode, the entire game board is redrawn on the server-side, encoded, and sent to the browser. You should find that the game responds slowly.

To run a game with a performance bottleneck:

1 In the Game Pack, click **Play** beside Minesweeper.

2 Select the **Method Time** option.

Note Clicking the link displays the option's definition.

3 In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.

4. In the Set Recording Level dialog box, set the slider to select **Full Encapsulation** and click **OK**.
5. In the Game Pack, click **Start** and click on several tiles until you are blocked.
6. Click **Quit**.
7. In the Runtime Summary view, click **Set Recording Level** on the toolbar.
8. In the Set Recording Level dialog box, set the slider to select **Data Recording Off**, then click **OK**.

   JProbe takes a snapshot and displays it in the Snapshots Navigator.
9. In the Snapshots Navigator, right-click this snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.
10. Name the snapshot *minesweeper_methodtime* and click **Save**.

    The new name is displayed in the Snapshots Navigator.
11. Click **Detach From Session** .

    The Summary view appears after a few seconds.

    Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.
12. On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Shutting Down the Server

In this step, you close the Geronimo-Tomcat application server.

To stop the Geronimo-Tomcat server:

1. Open a command window and navigate to:

   `<JPROBE_HOME>\demos\settings`
2. Type in the following command:
   - In Windows:

     `startWith_PerfAnalysis.bat stop`
   - In a UNIX or Linux shell:

     `startWith_PerfAnalysis.sh stop`
3. Type in Username: `system`.

4 Type in Password: `manager`.

The Geronimo-Tomcat application server shuts down and the JProbe Execution Console closes.

Step 5: Identifying the Performance Bottleneck

In this step, you use the Performance views to identify a hotspot. A hotspot is an expensive method, one that takes more time than necessary to run. The hotspot method may be the performance bottleneck or the method may call another method that causes the slowdown. In this example, you find two expensive third-party methods that are called by one of the Game Pack methods.

To identify the performance bottleneck:

1 In the Summary view, select one of the top methods suggested for investigation (`java.awt.MediaTracker.waitForID(int)`) in the Performance Hotspot section, right-click, and select **Investigate**.

The Methods view appears.

2 Select **Hot Methods (Elapsed Time)** from the *Color to Show* list.

Tip You can change the default Red-Gray color scheme. Right-click the color scale (located between the graph and the list) and select a different color scheme.

3 In the Call Graph, identify the critical node. In this example, it is the last node on the right side, `MediaTracker.waitForID()`.

4 Hover over the `MediaTracker.waitForID()` node to display its tooltip.

You can see that the Elapsed Method Time is 495 ms and it is called 600 times.

Note The actual number for time and calls for the function `waitForID()` can vary depending upon how many times the minesweeper image is being displayed.

Conclusion: The performance bottleneck is caused by the Game Pack method `renderMineMapImage_MethodTime` calling these expensive, third-party methods for every tile in the game board.

Step 6: Running the Session with Improved Code

After you discover the performance bottleneck, you can review the source code and choose the best way to fix your code. You could attempt to call a less expensive method or you might choose an entirely new algorithm to do the same task. For the Game Pack demo, we decided to use a different way to build the game board, one that does not require that the images be loaded and encoded on the server side. Instead, the game board is simply a table of HTML links. If image caching is enabled in the browser, the

images are stored and used locally; otherwise, the links point to images on the Web server.

To run the session with improved code:

1. Repeat Step 1: Starting the Session and the Game Pack to Step 4: Shutting Down the Server, selecting **Normal** mode instead of the **Method Time** option, and save the snapshot as *minesweeper_normal*.
2. Right-click *minesweeper_normal* and select the **Open Snapshot**.

   The Summary view appears.
3. Select the top method suggested for investigation (`demos.gamepack.minesweeper.MineSweeperRenderer.renderMineMapImage_Normal`) in the Performance Hotspot section, right-click, and select **Investigate**.

   The Methods view appears.
4. In the Call Graph, hover over the `renderMineMapImage_Normal` node to display its tooltip.

   You can see that the Elapsed Method Time is 23 ms and it is called 5 times. The new method `renderMineMapImage_Normal` runs faster than the original `renderMineMapImage_MethodTime` method. Also, the new method does not require the encoding servlet, so overall the program is even faster.

Step 7: Measuring the Performance Improvement

You know that the image caching algorithm runs much faster than the original compression algorithm. To quantify the improvement, use the Snapshot Difference tool to compare snapshots.

To measure the performance improvement:

1. In the JProbe Snapshots Navigator, right-click the *minesweeper_normal* snapshot and select **Snapshot Differencing**.

   The Performance Difference dialog box appears. The selected snapshot is displayed in the Snapshot to Compare field.
2. Select *minesweeper_methodtime* from the Baseline Snapshot list, and click **OK**.

   If you isolated on different methods, you see a message informing you that the snapshots have different transformations. The isolate action, among others, is removed automatically to ensure that you are comparing the same data set.

The Performance Difference view opens. By default, only the classes with differences are displayed.

3 To see the impact that changing the algorithm had on the servlet, isolate the `doGet` method by typing `*.doGet()` in the Filter Methods field.

The `GameController.doGet` and `GameImageServlet.doGet` methods are the only methods displayed. A negative number in the Elapsed Cumulative Time column means an improvement. The Normal mode offers a significant improvement over the Method Time mode.

This example demonstrates how, in your own code, an inefficient algorithm can significantly impact performance. Of course, you only want to optimize algorithms in the critical path of your program; there is no point tuning algorithms that are called very rarely. You also need to evaluate the overall impact on the runtime of the program. If an inefficient algorithm takes a total of a few seconds to execute, you may make it run faster, but the impact on the overall runtime of the program would be negligible.

Game Pack Coverage Analysis Tutorial

The Game Pack Coverage tutorial (Code Coverage Tutorial) describes how to assess the code coverage of your Game Pack program using the JProbe Coverage analysis tool and the pre-defined *Demo Gamepack Coverage Analysis* configuration.

Code Coverage Tutorial

This basic tutorial shows you how to evaluate the effectiveness of two test cases for the Game Pack Minesweeper game. In the first case, play the Minesweeper game in Normal mode. In the second case, play the Minesweeper game in Loitering Object mode. At the end, compare the results obtained during the two test cases and measure the code coverage.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo, Minesweeper Game
Test Case 1:	Run Minesweeper in Normal mode.
Test Case 2:	Run Minesweeper in Loitering Object mode.

The tutorial walks you through the following steps:

- Step 1: Starting the Session and the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Shutting Down the Server
- Step 5: Merging the Test Results
- Step 6: Assessing Your Test Coverage

Step 1: Starting the Session and the Game Pack

In this step, you start the Geronimo-Tomcat server using the pre-defined startup script shipped with JProbe, then connect JProbe to the Game Pack application running on this server.

To start the session from command line:

1 Type in the following command, depending on your platform:

- In Windows:

  ```
  <JPROBE_HOME>\demos\settings\startWith_CovAnalysis.bat
  ```

- In a UNIX or Linux shell:

  ```
  <JPROBE_HOME>\demos\settings\startWith_CovAnalysis.sh
  ```

  The Geronimo-Tomcat application server and the Game Pack application start in a command line window, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

  Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

2 Open the JProbe Console and click **Attach to Session**.

3 In the Attach to Session dialog box, select the JProbe Analysis Engine and click **OK**.

  JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

To start the session from the JProbe Console:

1 Click the **Configurations** tab and expand **Apache**.

  You can see that JProbe includes a pre-defined configuration file for the Game Pack coverage demo.

2 Click **Demo Gamepack Coverage Analysis**.

The Gamepack_Cov_Settings.jpl tab is displayed in the View area.

3 Click through the configuration tabs to review the settings.

Note These pre-defined settings cannot be modified or deleted. Also, you do not need to integrate your application with JProbe, because a startup script is already provided for this demo.

4 To run the Game Pack startup script and also connect the JProbe Console, click **Run & Connect**.

The JProbe Execution Console opens, the Geronimo-Tomcat application server and the Game Pack application start, and the *Welcome to the JProbe Game Pack Demo Application* page opens in a Web browser.

Note If the Web browser does not open the welcome page, you can open a browser and type: *http://localhost:8686/gamepack*. The default value of the pre-configured Geronimo HTTP port is 8686. The Game Pack welcome (startup) page appears.

JProbe connects to the Game Pack application, and the Coverage Session view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1 In the *Welcome to the JProbe Game Pack Demo Application* page, click **Start using Game Pack**.

The Game Pack demo login page appears.

2 Enter your **User ID** and **Password**, then click **Login**.

Note The first time you do this, you need to create a user name and password for yourself. For more information, see “Creating a User Account for Game Pack” on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

You are now ready to run a Coverage analysis session.

Step 3: Running the Session

In this step, you play the Minesweeper game in Normal mode three consecutive times (test case 1), then three more times in Object Loitering mode (test case 2), without quitting. It does not matter for the analysis whether you win or lose the games. For each test case, you take a snapshot.

To run the session for a coverage analysis:

1 Perform test case 1:

a In the Game Pack demo, click **Play** beside Minesweeper.

b Select the **Normal** option.

Note Clicking the link displays the option's definition.

c Click **Start** and play the game.

d When the game ends, play the game twice more (without quitting) for a total of three complete games.

e Click **Quit**.

f In the JProbe Console, click **Take Snapshot** on the toolbar.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

g In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

h Name the snapshot *coverage_normal* and click **Save**.

The snapshot name is displayed in the Snapshots Navigator.

2 Perform test case 2:

a In the Game Pack demo, click **Play** beside Minesweeper.

b Select the **Loitering Object** option.

Note Clicking the link displays the option's definition.

c Click **Start** and play the game.

d When the game ends, play the game twice more (without quitting) for a total of three complete games.

e Click **Quit**.

f In the JProbe Console, click **Take Snapshot** on the toolbar.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

g In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

h Name the snapshot *coverage_loitering* and click **Save**.

The snapshot name is displayed in the Snapshots Navigator.

3 Click **Detach From Session** .

The Snapshot Browser view appears after a few seconds.

4 Close the Coverage Session and the Snapshot Browser views.

5 On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Shutting Down the Server

In this step, you close the Geronimo-Tomcat application server.

To stop the Geronimo-Tomcat server:

1 Open a command window and navigate to:

```
<JPROBE_HOME>\demos\settings
```

2 Type in the following command:

- In Windows:

```
startWith_CovAnalysis.bat stop
```

- In a UNIX or Linux shell:

```
startWith_CovAnalysis.sh stop
```

3 Type in Username: `system`.

4 Type in Password: `manager`.

The Geronimo-Tomcat application server shuts down and the JProbe Execution Console closes.

Step 5: Merging the Test Results

In this step, you merge the snapshots taken during Step 3: Running the Session, to get a complete picture of the coverage provided by these two test cases.

You merge the snapshots using the `jpcovmerge` command line tool.

To merge the snapshots:

1 Click **Start** > **Run** and type the following, then click **OK**:

- In a Windows command window:

```
<JPROBE_HOME>\bin\jpcovmerge
<working_directory>\coverage_normal.jpc
<working_directory>\coverage_loitering.jpc
<working_directory>\coverage_merged.jpc
```

- In a UNIX or Linux shell:

```
<JPROBE_HOME>/bin/jpcovmerge
<working_directory>/coverage_normal.jpc
```

```
<working_directory>/coverage_loitering.jpc
<working_directory>/coverage_merged.jpc
```

Important This command fails if `<jprobe_home>` or `<working_directory>` has spaces in it. You must quote the application/arguments in this case.

2. On the JProbe Console, click **File > Open Snapshot**.
3. In the Open dialog box, select *<working_directory>\coverage_merged.jpc* and click **Open**.

   The merged snapshot is selected and its content is displayed in the Snapshot Browser view.

Step 6: Assessing Your Test Coverage

After merging your test cases, you are now ready to investigate your results. By default, JProbe displays results in terms of misses. A miss means that the code was not called during your test cases.

To investigate your results:

1. In the Snapshot Browser view, expand the class tree in the upper pane to show all of the classes in `demos.gamepack.minesweeper`, the game that was exercised.
2. Click the **% Missed Methods** column heading in the upper pane to sort the table from highest-to-lowest percentage of missed methods.

   Methods are missed in several classes: `MineSweeperException`, `MineSweeperModel`, `MineSweeperScore`, `MineSweeperSnapshot`, `MineSweeperMove`, and `MineSweeperRenderer`. We will investigate the first two classes.
3. In the upper pane, select the `MineSweeperModel` class.

   The methods for this class are displayed in the lower pane.
4. Select the `play()` method, right-click, and select **Show Source** to open the *<JPROBE_HOME>\demos\gamepack\src\demos\gamepack\minesweeper\MineSweeperModel.java* source code.

   The source code opens at a block of missed lines that are in an `if` statement, so you know that the test case missed a condition.
5. Click the **Down** button at the top of the coverage bar (to the right of the scroll bar) to move to the next set of missed lines.

6 Continue in this way until you have examined all the missed blocks of code, then move on to the next missed class.

Tip Once you identify an area for improvement, you can modify the test suite and redo the tutorial to see the improvement in overall coverage.

Advanced Game Pack Evaluation

This section describes how to deploy the JProbe Game Pack demo software on several third-party application servers supported by JProbe, and presents how to use JProbe with JavaEE applications that require these application servers.

For details, see the following topics:

- "Deploying the Game Pack Demo on Your Application Server" on page 95
- "Running Game Pack with the JProbe Application" on page 109
- "Game Pack Memory Analysis Tutorials" on page 110
- "Game Pack Performance Analysis Tutorial" on page 122
- "Game Pack Coverage Analysis Tutorial" on page 128

Deploying the Game Pack Demo on Your Application Server

This section presents how to deploy the Game Pack demo on the following application servers:

- "Deploying the Game Pack Demo on Apache Geronimo" on page 95
- "Deploying the Game Pack Demo on JBoss" on page 101
- "Deploying the Game Pack on BEA WebLogic Server 10.3" on page 103
- "Deploying the Game Pack on Oracle WebLogic Server 11g" on page 106

Deploying the Game Pack Demo on Apache Geronimo

The process of deploying the Game Pack demo on an Apache Geronimo application server, using the *Gamepack.ear* file for Geronimo provided with Game Pack, consists of the following main steps:

1 Creating a Database for the Game Pack Demo

2 Deploying Game Pack on Geronimo Using the Provided EAR File

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead. The process of deploying the Game Pack demo on an Apache Geronimo application server, using a customized EAR file, consists of the following main steps:

1. Creating a Database for the Game Pack Demo
2. Building and Deploying Game Pack on Geronimo.

Creating a Database for the Game Pack Demo

You must create a database before deploying Game Pack on an Apache Geronimo application server. The following procedure assumes that you have successfully installed a supported version of Apache Geronimo.

To create a database for the Game Pack demo:

1. Start the Apache Geronimo application server using the *<GERONIMO_HOME>\bin\startup.bat* file.
2. Open the Geronimo Console by typing *http://localhost:8080/console* in your browser window.
3. Log in to the Geronimo Console using your username and password. The default username/password is `system/manager`.
4. In the Console Navigation pane, click the **Embedded DB > DB Manager** link.
5. In the **Run SQL** pane, type in a name for the new database (for example, `GamepackDB`), and click **Create**.

The database is created and its name (*GamepackDB*) appears in the **DB Viewer** pane, in the Database List.

6 In the **Run SQL** pane, select **GamepackDB** from the *Use DB* list.

7 Using a text editor of your choice, open the *<JPROBE_HOME>\demos\gamepack\support\deployment\Geronimo\db\gamepack.sql* file provided with the demo, copy its content into the *SQL Commands* field in the Geronimo Console, then click **Run SQL**.

The GamepackDB table is now initialized and ready to use.

8 Log out of the Geronimo Console.

Deploying Game Pack on Geronimo Using the Provided EAR File

Use the build properties file and build script provided. The following procedure assumes that you have successfully installed and configured the Ant 1.6.3 or later, a supported version of Apache Geronimo, and JProbe v.8.3. In addition, you need to have defined an environment variable called `GERONIMO_HOME` which points to your Geronimo home directory. Also, you must have already created a GamepackDB database, as specified in section Creating a Database for the Game Pack Demo.

To deploy the Game Pack demo on Apache Geronimo:

1. In the <*JPROBE_HOME*>*demos**gamepack**build* directory, edit the *build.properties* file.
   a. Delete the pound sign (#) from the `J2EEServer=Geronimo` statement.
   b. Add a pound sign (#) in front of the other `J2EEServer` statements (that is, `J2EEServer=JBossWithTomcat` and `J2EEServer=WebLogic10.3`).
   c. Save the file.
2. If it is not already started, start the Apache Geronimo application server using the <*GERONIMO_HOME*>*bin**startup.bat* file.
3. Deploy the *Gamepack.ear* file provided by JProbe in the <*JPROBE_HOME*>*demos**gamepack**support**deployment**Geronimo* directory. Use one of the following methods:

   - **Method 1**: Open a command window, navigate to the <*JPROBE_HOME*>*demos**gamepack* directory and type the following command:

     ```
     ant -f build\build.xml deploy-ear
     ```

     Note To undeploy Game Pack, in a command window navigate to <*JPROBE_HOME*>*demos**gamepack* and type the following command:
     `ant -f build/build.xml undeploy`

   - **Method 2**: Open a command window, navigate to the <*GERONIMO_HOME*>*bin* directory and type the following command:

     ```
     deploy.bat --user system --password manager deploy
     <JPROBE_HOME>\demos\gamepack\support\deployment\Geronimo\
     Gamepack.ear
     ```

     Note To undeploy Game Pack, in a command window navigate to <GERONIMO_HOME>\bin and type the following command:
     `deploy.bat --user system --password manager undeploy gamepackear/gamepack-ear/1.0/ear`

   A successful build includes the following types of messages:

   ```
   Buildfile: build\build.xml

   deploy-ear:

   deploy-ear:

   deploy.Geronimo.error:

   deploy.Geronimo:

   deploy.toScript.error:
   ```

```
deploy.toScript:
     [exec] Using GERONIMO_HOME: <GERONIMO_HOME>
     [exec] Using GERONIMO_TMPDIR: var\temp
     [exec] Using JRE_HOME:        C:\Program
Files\Java\jdk1.6.0_10\jre
     [exec]     Deployed gamepackear/gamepack-ear/1.0/ear
     [exec]       `-> GamepackWeb.war @ /gamepack
     [exec]       `-> GamepackEJB.jar
     [exec]       `-> GamepackDataSource
     [echo] INFO: '<JPROBE_HOME>\demos\gamepack/dist/
gamepack.ear
' has been deployed to '<GERONIMO_HOME>' Geronimo Server.
BUILD SUCCESSFUL
```

4 To view the Game Pack, open a browser and type:

`http://localhost:<port>/gamepack`

where <*port*> is Geronimo's http port number. The default value is 8080.

The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

Building and Deploying Game Pack on Geronimo

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead.

To correctly build the Game Pack code with latest EJB 3.0 specification dependency libraries for JEE 1.5, the `classpath.J2EEServer` property must contain paths to the following JAR files existing in the Geronimo installation directory:

- *geronimo-ejb_3.0_spec-1.0.1.jar*
- *geronimo-jpa_3.0_spec-1.1.1.jar*
- *geronimo-jta_1.1_spec-1.1.1.jar*
- *geronimo-servlet_2.5_spec-1.2.jar*

Alternatively, you can set the `classpath.J2EEServer` property to point to the Geronimo *specs* repository.

To build and deploy the Game Pack demo on Geronimo:

1. In the <*JPROBE_HOME*>*demos**gamepack**build* directory, edit the *build.properties* file.

   a. Update `classpath.J2EEServer` property with the paths to the following files:

   ```
   <GERONIMO_HOME>\repository\org\apache\geronimo\specs\geron
   imo-ejb_3.0_spec\1.0.1\geronimo-ejb_3.0_spec-1.0.1.jar
   ```

   ```
   <GERONIMO_HOME>repository\org\apache\geronimo\specs\geroni
   mo-jpa_3.0_spec\1.1.1\geronimo-jpa_3.0_spec-1.1.1.jar
   ```

   ```
   <GERONIMO_HOME>\repository\org\apache\geronimo\specs\geron
   imo-jta_1.1_spec\1.1.1\geronimo-jta_1.1_spec-1.1.1.jar
   ```

   ```
   <GERONIMO_HOME>\repository\org\apache\geronimo\specs\geron
   imo-servlet_2.5_spec\1.2\geronimo-servlet_2.5_spec-1.2.jar
   ```

   Alternatively, you can set the `classpath.J2EEServer` property to point to the Geronimo *specs* repository:

   ```
   classpath.J2EEServer=<GERONIMO_HOME >/repository/org/
   apache/geronimo/specs/
   ```

   b. Delete the pound sign (`#`) from the `J2EEServer=Geronimo` statement.

   c. Add a pound sign (`#`) in front of the other `J2EEServer` statements (that is, `J2EEServer=JBossWithTomcat` and `J2EEServer=WebLogic10.3`).

   d. Save the file.

2. If it is not already started, start the Apache Geronimo application server using the <*GERONIMO_HOME*>*bin**startup.bat* file.

3. To build and deploy the Game Pack code, open a command window, navigate to the <*JPROBE_HOME*>*demos**gamepack* directory, and enter the following command:

   ```
   ant -f build\build.xml deploy
   ```

   The *Gamepack.ear* file is created in the <*JPROBE_HOME*>*demos**gamepack**dist* directory and the Game Pack application is deployed on the Geronimo application server.

4. To view the Game Pack, open a browser and type:

   ```
   http://localhost:<port>/gamepack
   ```

   where <*port*> is Geronimo's http port number. The default value is 8080.

The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

Deploying the Game Pack Demo on JBoss

For instructions on how to deploy the Game Pack demo on a JBoss application server, using the *Gamepack.ear* file for JBoss provided with Game Pack, see section Deploying Game Pack on JBoss Using the Provided EAR File.

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead. For instructions on how to deploy the Game Pack demo on a JBoss application server, using a customized EAR file, see section Building and Deploying Game Pack on JBoss.

Deploying Game Pack on JBoss Using the Provided EAR File

Use the build properties file and build script provided. The following procedure assumes that you have successfully installed and configured the Ant 1.6.3 or later, a supported version of JBoss, and JProbe v.8.3. In addition, you need to have defined an environment variable called `JBOSS_HOME` which points to your JBoss home directory.

To deploy the Game Pack demo on JBoss default server:

1. In the *<JPROBE_HOME>\demos\gamepack\build* directory, edit the *build.properties* file.
   a. Delete the pound sign (`#`) from the `J2EEServer=JBossWithTomcat` statement.
   b. Add a pound sign (`#`) in front of the other `J2EEServer` statements (that is, `J2EEServer=Geronimo` and `J2EEServer=WebLogic10.3`).
   c. Save the file.

   Note A `JBOSS_HOME` environment variable is required for the build to work.

2. At a command prompt, navigate to the *<JPROBE_HOME>\demos\gamepack* directory and enter the following command:

   Note The Ant `deploy-ear` command given in this procedure deploys on JBoss the Game Pack files provided with JProbe.

   ```
   ant -f build\build.xml deploy-ear
   ```

   A successful build includes the following types of messages:

   ```
   deploy.tofolder:
   ```

```
[copy] Copying 1 file to <JBOSS_HOME>\server\default\deploy

[echo] INFO: '<JPROBE_HOME>\demos\gamepack/support/
deployment/JBossWithTomcat/gamepack.ear

' has been deployed to '<JBOSS_HOME>\server/default/deploy'.

BUILD SUCCESSFUL
```

3. Start JBoss using the *<JBOSS_HOME>\bin\run.bat* file.
4. To view the Game Pack, open a browser and enter:

   `http://localhost:<port>/gamepack`

   where *<port>* is Tomcat's http port number. The default value is 8080.

   The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

Building and Deploying Game Pack on JBoss

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead.

To correctly build the Game Pack code with latest EJB 3.0 specification dependency libraries for JEE server, the `classpath.J2EEServer` property must contain paths to the following JAR files existing in the JBoss installation directory:

- *jboss-javaee.jar*
- *ejb3-persistence.jar*

Alternatively, you can copy the JBOSS Java EE 5 libraries (*jboss-javaee.jar* and *ejb3-persistence.jar*) in *<JBPROBE_HOME>/gamepack/lib* directory (no property setting is required in this case).

To build and deploy the Game Pack demo on JBoss:

1. In the *<JPROBE_HOME>\demos\gamepack\build* directory, edit the *build.properties* file.
   a. Update `classpath.J2EEServer` property with the paths to the following files:

      ```
      <JBOSS_HOME>/client/jboss-javaee.jar
      <JBOSS_HOME>/client/ejb3-persistence.jar
      ```

b Delete the pound sign (#) from the `J2EEServer=JBossWithTomcat` statement.

c Add a pound sign (#) in front of the other `J2EEServer` statements (that is, `J2EEServer=Geronimo` and `J2EEServer=WebLogic10.3`).

d Save the file.

2 At a command prompt, navigate to the *<JPROBE_HOME>\demos\gamepack* directory and enter the following command:

```
ant -f build\build.xml deploy
```

The *Gamepack.ear* file is created in the *<JPROBE_HOME>\demos\gamepack\dist* directory and the new Game Pack build is deployed on the JBoss application server.

A successful build includes the following types of messages:

```
deploy.tofolder:
[copy] Copying 1 file to <JBOSS_HOME>\server\default\deploy
[echo] INFO: '<JPROBE_HOME>\demos\gamepack/dist/gamepack.ear
' has been deployed to '<JBOSS_HOME>\server/default/deploy'.
BUILD SUCCESSFUL
```

Note To compile the Game Pack code, use the following command instead:

```
ant -f build\build.xml compile
```

3 Start JBoss using the *<JBOSS_HOME>\bin\run.bat* file.

4 To view the Game Pack, open a browser and enter:

```
http://localhost:<port>/gamepack
```

where *<port>* is Tomcat's http port number. The default value is 8080.

The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

Note To rebuild the Game Pack from the source and package and deploy it, use:

```
ant -f build\build.xml undeploy
```

Deploying the Game Pack on BEA WebLogic Server 10.3

For instructions on how to deploy the Game Pack demo on a BEA WebLogic Server 10.3 application server, using the *Gamepack.ear* file for BEA WebLogic Server 10.3

provided with Game Pack, see section Deploying Game Pack on BEA WebLogic Server 10.3 Using the Provided WebLogic Template JAR File.

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead. For instructions on how to deploy the Game Pack demo on a BEA WebLogic Server 10.3 application server, using a customized EAR file, see section Building and Deploying Game Pack on BEA WebLogic Server 10.3.

Deploying Game Pack on BEA WebLogic Server 10.3 Using the Provided WebLogic Template JAR File

To deploy the JProbe Game Pack on BEA WebLogic Server 10.3, you use the WebLogic Configuration Wizard. The following procedure assumes that you have successfully installed and configured BEA WebLogic Server 10.3 and JProbe v.8.3.

To deploy the Game Pack demo on BEA WebLogic Server 10.3:

1. Start the WebLogic Configuration Wizard.
   - On Windows, click **Start > Programs > BEA Products > Tools > Configuration Wizard**.
   - On UNIX systems, execute `java -jar configwiz.jar` in the *WL_HOME/wlserver_10.3/common/lib* directory.
2. Select the **Create a new WebLogic domain** check box and click **Next**.
3. Select the **Base this domain on an existing template** check box.
4. Click **Browse** and navigate to *<JPROBE_HOME>/demos/gamepack/support/deployment/Weblogic10.3/templates/domains* directory, select *gamepack.jar*, and click **OK**.
5. Click **Next**.
6. Configure the **User name** and **User password** and click **Next** (default is recommended).
7. Specify the server start mode by selecting the **Development Mode** check box.
8. Select a JDK from the available list (default is recommended), then click **Next**.
9. To use the default configuration (recommended), select the **No** check box.

   Note Alternatively, for customizing different configuration options, such as listen ports, select the **Yes** check box. This adds several steps to the configuration process.

10 Specify the directory where the WebLogic domain will be created. The default is in the *user_projects/domain* directory of your *WL_HOME*. The default domain name is *gamepack*. You can leave the default directory or change it.

11 Specify the directory where WebLogic Applications will be stored. The default is the *user_projects/applications* directory of your *WL_HOME*. You can leave the default directory or change it.

12 Click **Create** to create the domain in the selected directory.

13 When the configuration is created, click **Done**.

14 Start the Game Pack using your new configuration by executing the `startWebLogic.cmd` command from the *WL_HOME/user_projects/domains/gamepack* directory (or other directory if you made changes in the Configuration Wizard).

15 To view the Game Pack, open a browser and type:

`http://localhost:<port>/gamepack`

where <*port*> is WebLogic's http port number. The default value is `7001`.

The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

You can now stop WebLogic, by using the `stopWebLogic.cmd` command from the *WL_HOME/user_projects/domains/gamepack/bin* directory (or other directory if you made changes in the Configuration Wizard), and prepare to run the Game Pack in JProbe.

Building and Deploying Game Pack on BEA WebLogic Server 10.3

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead.

To correctly build the Game Pack code with latest EJB 3.0 specification dependency libraries for JEE server, the `classpath.J2EEServer` property must contain paths to the following JAR files existing in the BEA WebLogic Server 10.3 installation directory:

- *api.jar*
- *javax.persistence_1.0.0.0_1-0.jar*

Alternatively, you can copy the WebLogic Java EE 5 libraries (*javax.persistence_1.0.0.0_1-0.jar* and *api.jar*) in <*JBPROBE_HOME*>*/gamepack/lib* directory (no property setting is required in this case).

To build and deploy the Game Pack demo on BEA WebLogic Server 10.3:

1. In the <*JPROBE_HOME*>*demos**gamepack**build* directory, edit the *build.properties* file.
   a. Update `classpath.J2EEServer` property with the paths to the following files:

      `<WLS_HOME>/wlserver_10.3/server/lib/api.jar`

      `<WLS_HOME>/modules/javax.persistence_1.0.0.0_1-0.jar`

   b. Delete the pound sign (`#`) from the `J2EEServer=WebLogic10.3` statement.

   c. Add a pound sign (`#`) in front of the other `J2EEServer` statements (that is, `J2EEServer=Geronimo` and `J2EEServer=JBossWithTomcat`).

   d. Save the file.
2. At a command prompt, navigate to the <*JPROBE_HOME*>*demos**gamepack* directory and enter the following command:

   ```
   ant -f build\build.xml deploy
   ```

   The *Gamepack.ear* file is created in the <*JPROBE_HOME*>*demos**gamepack**dist* directory.

   Note WebLogic does not automatically deploy the new Gamepack.ear file. You must manually deploy it using the Weblogic Administration Console.
3. Use the WebLogic Administration Console to deploy the Gamepack.ear in the <*JPROBE_HOME*>*demos**gamepack**dist* directory.

Deploying the Game Pack on Oracle WebLogic Server 11g

For instructions on how to deploy the Game Pack demo on an Oracle WebLogic Server 11g application server, using the *Gamepack.ear* file for this server provided with Game Pack, see section Deploying Game Pack on Oracle WebLogic Server 11g Using the Provided WebLogic Template JAR File.

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead. For instructions on how to deploy the Game Pack demo on a Oracle WebLogic Server 11g application server, using a customized EAR file, see section Building and Deploying Game Pack on Oracle WebLogic Server 11g.

Deploying Game Pack on Oracle WebLogic Server 11g Using the Provided WebLogic Template JAR File

To deploy the JProbe Game Pack on Oracle WebLogic Server 11g, you use the WebLogic Configuration Wizard. The following procedure assumes that you have successfully installed and configured Oracle WebLogic Server 11g and JProbe v.8.3.

To deploy the Game Pack demo on Oracle WebLogic Server 11g:

1. Start the WebLogic Configuration Wizard.
   - On Windows, click **Start > Programs > Oracle Weblogic > WebLogic Server 11g > Tools > Configuration Wizard**.
   - On UNIX systems, execute `java -jar configwiz.jar` in the *WL_HOME/wlserver_10.3/common/lib* directory.
2. Select the **Create a new WebLogic domain** check box and click **Next**.
3. Select the **Base this domain on an existing template** check box.
4. Click **Browse** and navigate to *<JPROBE_HOME>/demos/gamepack/support/deployment/Weblogic10.3/templates/domains* directory, select *gamepack.jar*, and click **OK**.
5. Click **Next**.
6. Specify the directory where the WebLogic domain will be created. The default is in the *user_projects/domain* directory of your *WL_HOME*. The default domain name is *gamepack*. You can leave the default directory or change it.
7. Specify the directory where WebLogic Applications will be stored. The default is the *user_projects/applications* directory of your *WL_HOME*. You can leave the default directory or change it.
8. Configure the **User name** and **User password** and click **Next** (default is recommended).
9. Specify the server start mode by selecting the **Development Mode** check box.
10. Select a JDK from the available list (default is recommended), then click **Next**.

    The Configure JDBC Data Sources page appears, displaying the Game Pack data source name with host or user name information.
11. Do **not** select anything in the Data Source table in the Configure JDBC Data Sources page. Click **Next** without modifying any setting.

    The Configuring Data Source dialog box appears. After a few seconds, the following error message is displayed: `CFGFWK-60850: Test Failed!.`

12 Click **Next** without modifying any settings.

   A warning dialog box appears, displaying the following message: `Click Cancel and select Data Sources to test or Click OK to bypass testing`.

13 Click **OK**.

14 In the Select Optional Configuration page, review the default settings and click **Next**.

15 Click **Create** to create the domain in the selected directory.

16 When the configuration is created, click **Done**.

17 Start the Game Pack using your new configuration by executing the `startWebLogic.cmd` command from the *WL_HOME/user_projects/domains/gamepack* directory (or other directory if you made changes in the Configuration Wizard).

18 To view the Game Pack, open a browser and type:

   `http://localhost:<port>/gamepack`

   where <*port*> is WebLogic's http port number. The default value is `7001`.

   The Game Pack startup (welcome) page appears. You can now exercise the tutorials by clicking the available links. For more information, see "Running Game Pack with the JProbe Application" on page 109.

   You can now stop WebLogic, by using the `stopWebLogic.cmd` command from the *WL_HOME/user_projects/domains/gamepack/bin* directory (or other directory if you made changes in the Configuration Wizard), and prepare to run the Game Pack in JProbe.

Building and Deploying Game Pack on Oracle WebLogic Server 11g

If you want to make any changes in the Game Pack source files, then you have to build a new Game Pack EAR file using the updated code and deploy the new EAR file instead.

To correctly build the Game Pack code with latest EJB 3.0 specification dependency libraries for JEE server, the `classpath.J2EEServer` property must contain paths to the following JAR files existing in the Oracle WebLogic Server 11g installation directory:

- *api.jar*
- *javax.persistence_1.0.0.0_1-0-2.jar*

Alternatively, you can copy the WebLogic Java EE 5 libraries (*javax.persistence_1.0.0.0_1-0-2.jar* and *api.jar*) in *<JBPROBE_HOME>/gamepack/lib* directory (no property setting is required in this case).

To build and deploy the Game Pack demo on Oracle WebLogic Server 11g:

1. In the *<JPROBE_HOME>\demos\gamepack\build* directory, edit the *build.properties* file.
   a. Update `classpath.J2EEServer` property with the paths to the following files:

      `<WLS_HOME>/wlserver_10.3/server/lib/api.jar`

      `<WLS_HOME>/modules/javax.persistence_1.0.0.0_1-0-2.jar`

   b. Delete the pound sign (`#`) from the `J2EEServer=WebLogic10.3` statement.

   c. Add a pound sign (`#`) in front of the other `J2EEServer` statements (that is, `J2EEServer=Geronimo` and `J2EEServer=JBossWithTomcat`).

   d. Save the file.
2. At a command prompt, navigate to the *<JPROBE_HOME>\demos\gamepack* directory and enter the following command:

   ```
   ant -f build\build.xml deploy
   ```

   The *Gamepack.ear* file is created in the *<JPROBE_HOME>\demos\gamepack\dist* directory.

   Note WebLogic does not automatically deploy the new Gamepack.ear file. You must manually deploy it using the Weblogic Administration Console.
3. Use the WebLogic Administration Console to deploy the Gamepack.ear in the *<JPROBE_HOME>\demos\gamepack\dist* directory.

Running Game Pack with the JProbe Application

Follow a tutorial that matches the JProbe application you are using. You can open the tutorials from the Game Pack demo by clicking the **Tutorials** link.

To run Game Pack with JProbe, you must complete the following generic tasks:

1. Integrate the JProbe application with your application server.
2. Set up your JProbe session.
3. Run the JProbe session to start the application server.
4. Connect the JProbe Console to the session.

5 When the application server is started, open a browser and go to:

For Geronimo: *http://localhost:8080/gamepack*

For JBoss: *http://localhost:8080/gamepack*

For WebLogic Server: *http://localhost:7001/gamepack*

6 Exercise the use case.

Game Pack Memory Analysis Tutorials

The Game Pack Memory tutorials describe how to find two different kinds of memory leaks using the JProbe Memory analysis tool:

- "Loitering Objects Tutorial" on page 110
- "Object Cycling Tutorial" on page 116

Loitering Objects Tutorial

Minesweeper and the Match Game both offer a Loitering Objects mode. The Loitering Objects mode demonstrates how an obsolete container reference can keep session beans in the Java heap long after their usefulness is gone. In this tutorial, you see how you can reclaim memory by removing loitering `GameHandler` objects.

Before beginning this tutorial, you need to deploy the Game Pack to the application server of your choice. For more information, see "Deploying the Game Pack Demo on Your Application Server" on page 95.

Note The values cited in this tutorial reflect the Game Pack running on the JBoss 5.1.0 application server, on Windows XP.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack demo: Match Game
Use Case:	Play a few games without quitting.
Architecture:	When the Start button is selected, create a `GameHandler` object to run the game.

Hypothesis: The `GameHandler` object is removed from the heap when the game ends.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Session and Starting the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Identifying Loitering Objects
- Step 5: Investigating Loitering Objects
- Step 6: Running the Session with Improved Code

Step 1: Setting Up the Session and Starting the Game Pack

You use the JProbe Console to create the session settings for this example. In the setup procedure, only the settings that you need to change or verify are mentioned. If a setting is not mentioned, leave it blank or in its default state. The following procedure assumes that you are running JProbe locally on your computer.

To set up the session and start Game Pack:

1. On the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.

2. Click **JBoss > JBoss v5.x**.
3. Provide a name for this configuration in the Configuration Name text box (for example, *LoiteringObjects*), then click **Next**.
4. Type the path to your server startup script in the text field or click the browse button and navigate to it, then click **Next**.
5. In the My Code tab, in the Elements area, type the path to the deployed *gamepack.ear* file (for example, *<jprobe_home>\demos\gamepack\support\deployment\JBossWithTomcat\gamepack.ear*) or click the browse button and navigate to it.
6. Click **Create Filters**.

   The table is populated with the filters available for your application.

7. Select the My Application Filters check box.

All application filters are now selected and Action set to **include**. JProbe can use these application filters as default data collection filters when performing a Memory analysis.

8 Specify a Category/Program Name, then click **Next**.

9 In the Analysis Type tab, select **Memory Analysis**, then click **Next**.

10 Click **Next** again to pass the Advanced JProbe Options tab.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.

11 To create a JPL file and a startup script, click **Save & Integrate**.

12 Click **Save** to save the configuration file (*LoiteringObjects_Mem_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

13 In the Integrate tab, click the browse button and navigate to the location where you want to save the startup script (for example, in Windows: *run_WithJProbe.bat*, and in UNIX/Linux systems: *run_WithJProbe.sh*), then click **Save**.

Tip It is a good practice to save the script files into a directory which is recommended by the integration Status dialog, (for example, in this case, the recommended path is *C*:\<*jboss_home*>*bin* directory).

14 When the integration is complete, click **Next**.

The Run Configuration tab is displayed.

15 Select the **Close syscon window on exit** check box to automatically close the *jpsyscon* window after the termination of the launched script, if the program exits normally.

16 To run your application with JProbe using the startup script, click **Run & Connect**.

Note To start JBoss using the startup script:

- In Windows: C:\<jboss_home>\bin\run_WithJProbe.bat
- In a UNIX or Linux shell, from the directory where you saved the startup script: ./run_WithJProbe.sh

JProbe Execution Console opens, and the application server and the Game Pack application start. JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1. Open a Web browser and go to *http://localhost:8080/gamepack*.

   The Game Pack Demo login page appears.

2. Enter your **User ID** and **Password**, then click **Login**.

   Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

   You are now ready to run a Memory analysis session.

Step 3: Running the Session

In this step, you work through a use case by playing three consecutive games. It does not matter for the analysis whether you win or lose the games. However, it is important that you start a session before you begin to play, or JProbe will not perform a garbage collection. After you have played three games, you end the session, and JProbe takes a snapshot.

To run a game with the Loitering Objects fault:

1. In the Game Pack demo, click **Play** beside either Minesweeper or Match Game.

   Note Both games have the same loitering object problem.

2. Select the **Loitering Objects** option.

   Note Clicking the link displays the option's definition.

3. In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.

4. In the Set Recording Level dialog box, select **Record Allocations and Stack Traces For All Instances**, then click **OK**.

5. In the Game Pack demo, click **Start** and play the game.

6. When the game ends, play the game twice more (without quitting) for a total of three complete games.

7. Click **Quit**.

8 In the Runtime Summary view, click **Set Recording Level** on the toolbar.

9 In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

10 In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

11 Name the snapshot *loitering_objects* and click **Save**.

The snapshot name is displayed in the Snapshots Navigator.

12 Click **Detach From Session** .

The Instances view appears after a few seconds.

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

13 On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Identifying Loitering Objects

In this step, you look for loitering objects in the heap. The Heap Count column is the first place to look. In general, you should expect objects created during a session to be removed at the end of it. In fact, the `GameHandler` objects are not removed, and three instances of this object continue to loiter in the heap.

To identify loitering objects:

1 If the *loitering_objects* snapshot is not open, right-click it in the snapshot navigator and select **Open Instances View**.

The Instances view appears.

2 In the Filter Classes field, type `*.GameHandler` and press **Enter** to display only the `GameHandler` class.

The Heap Count for GameHandler is 3, not 0 (zero) as hypothesized.

Step 5: Investigating Loitering Objects

In this step, you find the live object that continues to hold a reference to the loitering instances of `GameHandler` in the heap. You start in the Instances view, then drill down to Instance Detail and the Memory Leak views, and discover that the loitering

`GameHandler` objects are being held by an obsolete container reference. To look for a solution, you open the Source view and review the code for the allocating method.

To investigate loitering objects:

1. If the snapshot is not open, right-click it in the snapshot navigator and select the **Open Instances View**.

   The Instances view appears.

2. Select **Heap Count** from the *Investigate by* list.

3. Filter the method list by typing `*.GameHandler` in the Filter Classes field and pressing **Enter**.

4. Select the `GameHandler` class and click **Instance Detail** .

   The Instance Detail view opens. The three loitering `GameHandler` objects are displayed in the instances list. You can see the stack trace of method calls in the **Allocated At** column.

5. Click the Trace tab and scroll down the method list to the methods belonging to the subpackages of the `demos.gamepack.web.game` package and find a method called `EJBControllerImpl.startGame()`.

   The `startGame()` method calls the `getGameHandlerRemote()` method, which in turn sets off a series of calls to JBoss methods that eventually causes the bean to be created. Therefore, the `getGameHandlerRemote()` method is the most likely candidate for investigation, because it is the last Game Pack method before the series of application server calls.

6. Select the `demos.gamepack.web.game.EJBControllerImpl.getGameHandlerRemote()` method, right-click, and select **Show Allocated At Source**.

7. If you are prompted for the source code, navigate to *<jprobe_home>/demos/gamepack/src/demos/gamepack/web/game*, select *EJBControllerImpl.java*, and click **Open**.

   The Source view opens, highlighting line 174 of the code. This line represents the method call from the `getGameHandlerRemote()` method to the application server's `create` method, which creates the beans.

   Now that you found the method that creates the loitering `GameHandler` objects, you are close to finding out where the objects should be removed. Scrolling down to line 185 you find that the `resetGameHandlerRemote()` method is the problem method. When the Loitering Objects mode is selected, the

`_gameHandlerRemote.removeObject()` method is not called, so the references to the GameHandler objects are never removed.

Tip It is good programming practice to pair your calls to create and remove objects close together.

Step 6: Running the Session with Improved Code

You can re-run Minesweeper or Match Game in Normal mode with the corrected code. Repeat Step 1: Setting Up the Session and Starting the Game Pack through Step 3: Running the Session, selecting **Normal** mode instead of **Loitering Objects** and not re-naming the snapshot. You will see in the Instances view that the Heap Count for GameHandler is now 0, as predicted by the hypothesis.

This example demonstrates how to use JProbe to identify and remove loitering objects from your code.

Object Cycling Tutorial

The Match Game offers an Object Cycling mode. The Object Cycling mode demonstrates how over-allocating short-lived objects can cause the garbage collector to run more frequently than necessary. Garbage collection takes time and resources. The Normal mode shows that it is often more efficient to reuse strings.

Before beginning this tutorial, you need to deploy the Game Pack to the application server of your choice. For more information, see “Deploying the Game Pack Demo on Your Application Server” on page 95.

Note The values cited in this tutorial reflect the Game Pack running on the JBoss 5.1.0 application server, on Windows XP.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo: Match Game
Use Case:	Play three games without quitting.
Architecture:	**Object Cycling**: Use string concatenation. **Normal**: Append to an existing string buffer object.

Hypothesis: The program does not create unnecessary temporary string objects.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Session and Starting the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Identifying Object Cycling
- Step 5: Investigating Object Cycling
- Step 6: Running the Session with Improved Code

Step 1: Setting Up the Session and Starting the Game Pack

You use the JProbe configuration tool to create the session settings for this example. In the setup procedure, only the settings that you need to change or verify are mentioned. If a setting is not mentioned, leave it blank or in its default state. The following procedure assumes that you are running JProbe locally on your computer.

To set up the session and start Game Pack:

1. On the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.
2. Click **JBoss > JBoss v5.x**.
3. Provide a name for this configuration in the Configuration Name text box (for example, *ObjectCycling*), then click **Next**.
4. Type the path to your server startup script in the text field or click the browse button … and navigate to it, then click **Next**.
5. In the My Code tab, in the Elements area, type the path to the deployed *gamepack.ear* file (for example, *<jprobe_home>\demos\gamepack\support\deployment\JBossWithTomcat\gamepack.ear*) or click the browse button … and navigate to it.
6. Click **Create Filters**.

   The table is populated with the filters available for your application.
7. Select the My Application Filters check box.

All application filters are now selected and Action set to **include**. JProbe can use these application filters as default data collection filters when performing a Memory analysis.

8 Specify a Category/Program Name, then click **Next**.

9 In the Analysis Type tab, select **Memory Analysis**, then click **Next**.

10 Click **Next** again to pass the Advanced JProbe Options tab.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.

11 To create a JPL file and a startup script, click **Save & Integrate**.

12 Click **Save** to save the configuration file (*ObjectCycling_Mem_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

13 In the Integrate tab, click the browse button and navigate to the location where you want to save the startup script (for example, in Windows: *run_WithJProbe.bat*, and in UNIX/Linux systems: *run_WithJProbe.sh*), then click **Save**.

Tip It is a good practice to save the script files into a directory which is recommended by the integration Status dialog, (for example, in this case, the recommended path is *C:\<jboss_home>\bin* directory).

14 When the integration is complete, click **Next**.

The Run Configuration tab is displayed.

15 Select the **Close syscon window on exit** check box to automatically close the *jpsyscon* window after the termination of the launched script, if the program exits normally.

16 To run your application with JProbe using the startup script, click **Run & Connect**.

Note To start JBoss using the startup script:

- In Windows: C:\<jboss_home>\bin\run_WithJProbe.bat
- In a UNIX or Linux shell, from the directory where you saved the startup script: ./run_WithJProbe.sh

JProbe Execution Console opens, and the application server and the Game Pack application start. JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1. Open a Web browser and go to *http://localhost:8080/gamepack*.

   The Game Pack Demo login page appears.

2. Enter your **User ID** and **Password**, then click **Login**.

   Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

   You are now ready to run a Memory analysis session.

Step 3: Running the Session

In this step, you turn on garbage monitoring so that you can see how many objects are garbage collected during your use case. You work through a use case by playing three consecutive games. It does not matter for the analysis whether you win or lose the games. However, it is important that you start a use case before you begin to play, or JProbe will not perform a garbage collection. After you have played three games, you end the use case, and JProbe takes a snapshot.

To run a game with the Object Cycling fault:

1. In the Game Pack demo, select **Play** beside Match Game.
2. Select the **Object Cycling** option.

   Note Clicking the link displays the option's definition.

3. In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.
4. In the Set Recording Level dialog box, select **Record Allocations, Stack Traces, and Garbage Data For All Instances** and click **OK**.
5. In the Game Pack demo, click **Start** and play the game.
6. When the game ends, play the game twice more (without quitting) for a total of three complete games.
7. Click **Quit**.

8 In the Runtime Summary view, click **Set Recording Level** on the toolbar.

9 In the Set Recording Level dialog box, select **Data Recording Off**, then click **OK**.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

10 In the Snapshots Navigator, right-click this snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

11 Name the snapshot *object_cycling* and click **Save**.

The new name is displayed in the Snapshots Navigator.

12 Click **Detach From Session** .

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

13 On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Identifying Object Cycling

In this step, you look for classes and methods that allocate short-lived objects.

The Garbage Collections chart in the GC Data tab displays steep spikes, which means that some set of objects is garbage collected soon after being created. In the Instances view, look for classes with high Dead Count values and no or very few instances still alive. In this example, you can see that many instances of `StringBuffer` were allocated and garbage collected. None of the instances are still alive. When you review the results, you see that most of the `StringBuffer` objects were allocated by `*_StringConcatenation` methods in the `MatchGameRenderer` class.

To identify short-lived objects:

1 If the *object_cycling* snapshot is not open, right-click it in the Snapshots Navigator and select **Open Instances View**.

The Instances view appears.

2 Select **Dead Count** from the *Investigate by* list, and sort the table by **Dead Count**.

You can see that many `String` and `StringBuffer` instances are immediately garbage collected (that is, high **Dead Count** values and low **Recorded Count** values).

3. Drill into the Merged Allocation Points and Call Traces views, by selecting the `StringBuffer` instance, right-clicking, and selecting the **Open Merged Allocation Points View** and **Open Call Traces View**, respectively.
4. In the Merged Allocation Points view, right-click My Code and select **Replace Category with More Detail**, then sort by Cumulative Count.

   You can see that most of the `StringBuffer` objects were allocated by `renderGameMap_StringConcatenation`, `renderGamePlay_StringConcatenation`, and `renderSnapshot_StringConcatenation` methods in the `MatchGameRenderer` class, `demos.gamepack.matchgame` package.

Step 5: Investigating Object Cycling

In Java, the JVM converts string concatenations into `StringBuffer` objects, which means that each concatenation creates a new object with a very short life span. This is a less efficient way to handle strings.

Take a look at the source code to see exactly how the application works. The `_StringConcatenation` methods are in the `MatchGameRenderer.java` source code. You can use any editor to examine the allocating methods, but it makes it easier to find the method if your editor has a search feature. Remember that in this case the code contains the fixed code as well.

To investigate the garbage collected objects:

1. Navigate to *<jprobe_home>/demos/gamepack/src/demos/gamepack/matchgame* and open the *MatchGameRenderer.java* file in a source code editor.
2. Search for the top method: `renderGamePlay_StringConcatenation`.
3. Observe that the method contains many string concatenations. If you scroll down to find `renderSnapshot_StringConcatenation` and then `renderGameMap_StringConcatenation`, you can see that these methods also use string concatenation.

Step 6: Running the Session with Improved Code

The file *MatchGameRenderer.java* also contains methods that offer a better way to handle the strings. While you have the source code open, you can scroll to find these improved methods:

- `renderGamePlay_StringBufferAppend`
- `renderSnapshot_StringBufferAppend`

- `renderGameMap_StringBufferAppend`

If you want, you can re-run this tutorial using **Normal** mode, which uses the improved methods. When you check the Instances view, you should see that the number of short-lived `StringBuffer` objects is reduced significantly.

This example demonstrates how to use JProbe to identify and remove object cycling problems in your code.

Game Pack Performance Analysis Tutorial

The Game Pack Performance tutorial (Performance Bottleneck Tutorial) describes how to investigate a performance bottleneck using the JProbe Performance analysis tool.

Performance Bottleneck Tutorial

The Minesweeper game can be played in the Method Time mode. This mode demonstrates a performance bottleneck caused by an inappropriate algorithm that is used for rendering the Minesweeper game board as one large image. For comparison, you can play the game in the Normal mode, which creates the game board as a table containing HTML links to images.

Before beginning this tutorial, you need to deploy the Game Pack to the application server of your choice. For more information, see “Deploying the Game Pack Demo on Your Application Server” on page 95.

Note The values cited in this tutorial reflect the Game Pack running on the JBoss 5.1.0 application server, on Windows XP.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo, Minesweeper Game Entry point of interest: `GameController.doGet`
Use Case:	Run minesweeper and select one tile.

Architecture:	**Method Time**: The game board is constructed with images on the server-side, encoded as a single image, and sent to the browser. You may notice the image jumps when a tile is selected. **Normal**: The game board is created as a table of links to images.
Hypothesis:	Encoding is slow. Creating the game board with links to images will be faster and smoother, especially if the images are cached by the browser.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Session and Starting the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Identifying the Performance Bottleneck
- Step 5: Running the Session with Improved Code
- Step 6: Measuring the Performance Improvement

Step 1: Setting Up the Session and Starting the Game Pack

You use the JProbe configuration wizard to create the session settings for this example. In the setup procedure, only the settings that you need to change or verify are mentioned. If a setting is not mentioned, leave it blank or in its default state. The following procedure assumes that you are running JProbe locally on your computer.

To set up the session and start the Game Pack:

1. On the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.
2. Click **JBoss > JBoss v5.x**.
3. Provide a name for this configuration in the Configuration Name text box (for example, *PerformanceBottleneck*), then click **Next**.
4. Type the path to your server startup script in the text field or click the browse button and navigate to it, then click **Next**.
5. In the My Code tab, in the Elements area, type the path to the deployed *gamepack.ear* file (for example, *<jprobe_home>\demos\gamepack*

support\deployment\JBossWithTomcat\gamepack.ear) or click the browse button ... and navigate to it.

6 Click **Create Filters**.

The table is populated with the filters available for your application.

7 Select the My Application Filters check box.

All application filters are now selected and Action set to **include**. JProbe can use these application filters as default data collection filters when performing a Performance analysis.

8 Specify a Category/Program Name, then click **Next**.

9 In the Analysis Type tab, select **Performance Analysis**.

10 In the Filters tab, ensure that all collection filters are selected and set Action to **line** for each of them, then click **Next**.

11 Click **Next** again to pass the Advanced JProbe Options tab.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.

12 To create a JPL file and a startup script, click **Save & Integrate**.

13 Click **Save** to save the configuration file (*PerformanceBottleneck_Perf_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

14 In the Integrate tab, click the browse button ... and navigate to the location where you want to save the startup script (for example, in Windows: *run_WithJProbe.bat*, and in UNIX/Linux systems: *run_WithJProbe.sh*), then click **Save**.

Tip It is a good practice to save the script files into a directory which is recommended by the integration Status dialog, (for example, in this case, the recommended path is *C:\<jboss_home>\bin* directory).

15 When the integration is complete, click **Next**.

The Run Configuration tab is displayed.

16 Select the **Close syscon window on exit** check box to automatically close the *jpsyscon* window after the termination of the launched script, if the program exits normally.

17 To run your application with JProbe using the startup script, click **Run & Connect**.

Note To start JBoss using the startup script:

- In Windows: C:\<jboss_home>\bin\run_WithJProbe.bat
- In a UNIX or Linux shell, from the directory where you saved the startup script: ./run_WithJProbe.sh

JProbe Execution Console opens, and the application server and the Game Pack application start. JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1 Open a Web browser and go to *http://localhost:8080/gamepack*.

The Game Pack Demo login page appears.

2 Enter your **User ID** and **Password**, then click **Login**.

Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

You are now ready to run a Performance analysis session.

Step 3: Running the Session

In this step, you exercise the use case by selecting one tile. It does not matter for the analysis whether the tile reveals a number or a mine. In Method Time mode, the entire game board is redrawn on the server-side, encoded, and sent to the browser. You should find that the game responds slowly.

To run a game with a performance bottleneck:

1 In the Game Pack, click **Play** beside Minesweeper.

2 Select the **Method Time** option.

Note Clicking the link displays the option's definition.

3 In the JProbe Runtime Summary view, click **Set Recording Level** on the toolbar.

4 In the Set Recording Level dialog box, set the slider to select **Full Encapsulation** and click **OK**.

5 In the Game Pack, click **Start** and click on several tiles until you are blocked.

6 Click **Quit**.

7 In the Runtime Summary view, click **Set Recording Level** on the toolbar.

8 In the Set Recording Level dialog box, set the slider to select **Data Recording Off**, then click **OK**.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

9 In the Snapshots Navigator, right-click this snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

10 Name the snapshot *minesweeper_methodtime* and click **Save**.

The new name is displayed in the Snapshots Navigator.

11 Click **Detach From Session** .

The Summary view appears after a few seconds.

Note If a different snapshot is already opened in the JProbe Console, the new snapshot taken does not open automatically. To open it, right-click the snapshot name in the Snapshot navigator and select **Open Snapshot**.

12 On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Identifying the Performance Bottleneck

In this tutorial, you use the Performance views to identify a hotspot. A hotspot is an expensive method, one that takes more time than necessary to run. The hotspot method may be the performance bottleneck or the method may call another method that causes the slowdown. In this example, you find two expensive third-party methods that are called by one of the Game Pack methods.

To identify the performance bottleneck:

1 In the Summary view, select the top method suggested for investigation (`java.awt.MediaTracker.waitForID(int)`) in the Performance Hotspot section, right-click, and select **Investigate**.

The Methods view appears.

2 Select **Hot Methods (Elapsed Time)** from the *Color to Show* list.

Tip You can change the default Red-Gray color scheme. Right-click the color scale (located between the graph and the list) and select a different color scheme.

3. In the Call Graph, identify the critical node. In this example, it is the last node on the right side, `MediaTracker.waitForID()`.
4. Hover over the `MediaTracker.waitForID()` node to display its tooltip.

   You can see that the Elapsed Method Time is 495 ms and it is called 600 times.

   Note The actual number for time and calls for the function `waitForID()` can vary depending upon how many times the minesweeper image is being displayed.

Conclusion: The performance bottleneck is caused by the Game Pack method `renderMineMapImage_MethodTime` calling these expensive, third-party methods for every tile in the game board.

Step 5: Running the Session with Improved Code

After you discover the performance bottleneck, you can choose how best to fix your code. Review the source code. You could attempt to call a less expensive method or you might choose an entirely new algorithm to do the same task. For the Game Pack demo, we decided to use a different way to build the game board, one that does not require that the images be loaded and encoded on the server side. Instead, the game board is simply a table of HTML links. If image caching is enabled in the browser, the images are stored and used locally; otherwise, the links point to images on the Web server.

To run the session with improved code:

1. Repeat Step 1: Setting Up the Session and Starting the Game Pack to Step 3: Running the Session, selecting **Normal** mode instead of the **Method Time** option, and save the snapshot as *minesweeper_normal*.
2. Right-click *minesweeper_normal* and select the **Open Snapshot**.

   The Summary view appears.
3. Select the top method suggested for investigation (`demos.gamepack.minesweeper.MineSweeperRenderer.renderMineMapImage_Normal`) in the Performance Hotspot section, right-click, and select **Investigate**.

   The Methods view appears.
4. In the Call Graph, hover over the `renderMineMapImage_Normal` node to display its tooltip.

   You can see that the Elapsed Method Time is 23 ms and it is called five times. The new method `renderMineMapImage_Normal` runs faster than the original `renderMineMapImage_MethodTime` method. Also, the new method does not require the encoding servlet, so overall the program is even faster.

Step 6: Measuring the Performance Improvement

You know that the image caching algorithm runs much faster than the original compression algorithm. To quantify the improvement, use the Snapshot Difference tool to compare snapshots.

To measure the performance improvement:

1. In the JProbe Snapshot Navigator, right-click the *minesweeper_normal* snapshot and select **Snapshot Differencing**.

   The Performance Difference dialog box appears. The selected snapshot is displayed in the Snapshot to Compare field.

2. Select *minesweeper_methodtime* from the Baseline Snapshot list, and click **OK**.

   If you isolated on different methods, you see a message informing you that the snapshots have different transformations. The isolate action, among others, is removed automatically to ensure that you are comparing the same data set.

   The Performance Difference view opens. By default, only the classes with differences are displayed.

3. To see the impact that changing the algorithm had on the servlet, isolate the `doGet` method by typing `*.doGet()` in the Filter Methods field.

   The `GameController.doGet` and `GameImageServlet.doGet` methods are the only methods displayed. A negative number in the Elapsed Cumulative Time column means an improvement. The Normal mode offers a significant improvement over the Method Time mode.

This example demonstrates how, in your own code, an inefficient algorithm can significantly impact performance. Of course, you only want to optimize algorithms in the critical path of your program; there is no point tuning algorithms that are called very rarely. You also need to evaluate the overall impact on the runtime of the program. If an inefficient algorithm takes a total of a few seconds to execute, you may make it run faster, but the impact on the overall runtime of the program would be negligible.

Game Pack Coverage Analysis Tutorial

The Game Pack Coverage tutorial (Code Coverage Tutorial) describes how to assess the code coverage of your Game Pack program using the JProbe Coverage analysis tool.

Code Coverage Tutorial

This basic tutorial shows you how to evaluate the effectiveness of two test cases for the Game Pack Minesweeper game. In the first case, play the Minesweeper game in Normal mode. In the second case, play the Minesweeper game in Loitering Object mode. At the end, compare the results obtained during the two test cases and measure the code coverage.

The following table summarizes the types of information you need to know before starting this tutorial.

Program:	Game Pack Demo, Minesweeper Game
Test Case 1:	Run Minesweeper in Normal mode.
Test Case 2:	Run Minesweeper in Loitering Object mode.

The tutorial walks you through the following steps:

- Step 1: Setting Up the Session and Starting the Game Pack
- Step 2: Logging in to Game Pack
- Step 3: Running the Session
- Step 4: Merging the Test Results
- Step 5: Assessing Your Test Coverage

Step 1: Setting Up the Session and Starting the Game Pack

You use the JProbe configuration wizard to create the session settings for this example. In the setup procedure, only the settings that you need to change or verify are mentioned. If a setting is not mentioned, leave it blank or in its default state. The following procedure assumes that you are running JProbe locally on your computer.

To set up the session and start the Game Pack:

1. On the JProbe Console, click **Configurations > New Configuration**.

   The New Configuration Wizard dialog box appears.

2. Click **JBoss > JBoss v5.x**.

3. Provide a name for this configuration in the Configuration Name text box (for example, *CoverageLoitering*), then click **Next**.

4 Type the path to your server startup script in the text field or click the browse button ... and navigate to it, then click **Next**.

5 In the My Code tab, in the Elements area, type the path to the deployed *gamepack.ear* file (for example, *<jprobe_home>\demos\gamepack\support\deployment\JBossWithTomcat\gamepack.ear*) or click the browse button ... and navigate to it.

6 Click **Create Filters**.

The table is populated with the filters available for your application.

7 Select the My Application Filters check box.

All application filters are now selected and Action set to **include**. JProbe can use these application filters as default data collection filters when performing a Memory analysis.

8 Specify a Category/Program Name, then click **Next**.

9 In the Analysis Type tab, select **Coverage Analysis**.

10 Click **Next** again to pass the Advanced JProbe Options tab.

The Summary tab is displayed. It presents a summary of the settings defined for your configuration.

11 To create a JPL file and a startup script, click **Save & Integrate**.

12 Click **Save** to save the configuration file (*CoverageLoitering_Cov_Settings.jpl*) into your working directory.

Tip It is a good practice to save the configuration files into a directory other than the JProbe installation directory (for example, *C:/JProbeFiles/*).

The Integration Wizard dialog box appears, displaying the integration status.

13 In the Integrate tab, click the browse button ... and navigate to the location where you want to save the startup script (for example, in Windows: *run_WithJProbe.bat*, and in UNIX/Linux systems: *run_WithJProbe.sh*), then click **Save**.

Tip It is a good practice to save the script files into a directory which is recommended by the integration Status dialog, (for example, in this case, the recommended path is *C:\<jboss_home>\bin* directory).

14 When the integration is complete, click **Next**.

The Run Configuration tab is displayed.

15 Select the **Close syscon window on exit** check box to automatically close the *jpsyscon* window after the termination of the launched script, if the program exits normally.

16 To run your application with JProbe using the startup script, click **Run & Connect**.

Note To start JBoss using the startup script:

- In Windows: C:\<jboss_home>\bin\run_WithJProbe.bat
- In a UNIX or Linux shell, from the directory where you saved the startup script: ./run_WithJProbe.sh

JProbe Execution Console opens, and the application server and the Game Pack application start. JProbe connects to the Game Pack application, and the Runtime Summary view opens in the Console.

Step 2: Logging in to Game Pack

In this step, you log in to the Game Pack demo.

1 Open a Web browser and go to *http://localhost:8080/gamepack*.

The Game Pack Demo login page appears.

2 Enter your **User ID** and **Password**, then click **Login**.

Note The first time you do this, you need to create a user name and password for yourself. For more information, see "Creating a User Account for Game Pack" on page 134. Alternatively, you can use the default user ID/password **admin/admin**.

You are now ready to run a Performance analysis session.

Step 3: Running the Session

In this step, you play the Minesweeper game in Normal mode three consecutive times (test case 1), then three more times in Object Loitering mode (test case 2), without quitting. It does not matter for the analysis whether you win or lose the games. For each test case, you take a snapshot.

To run the session for a coverage analysis:

1 Perform test case 1:

a In the Game Pack demo, click **Play** beside Minesweeper.

b Select the **Normal** option.

Note Clicking the link displays the option's definition.

c Click **Start** and play the game.

d When the game ends, play the game twice more (without quitting) for a total of three complete games.

e Click **Quit**.

f In the JProbe Console, click **Take Snapshot** on the toolbar.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

g In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

h Name the snapshot *coverage_normal* and click **Save**.

The snapshot name is displayed in the Snapshots Navigator.

2 Perform test case 2:

a In the Game Pack demo, click **Play** beside Minesweeper.

b Select the **Loitering Object** option.

Note Clicking the link displays the option's definition.

c Click **Start** and play the game.

d When the game ends, play the game twice more (without quitting) for a total of three complete games.

e Click **Quit**.

f In the JProbe Console, click **Take Snapshot** on the toolbar.

JProbe takes a snapshot and displays it in the Snapshots Navigator.

g In the Snapshots Navigator, right-click the snapshot, select **Save Snapshot As**, and navigate to where you want to save the snapshot.

h Name the snapshot *coverage_loitering* and click **Save**.

The snapshot name is displayed in the Snapshots Navigator.

3 Click **Detach From Session** .

The Snapshot Browser view appears after a few seconds.

4 Close the Coverage Session and the Snapshot Browser views.

5 On the Game Pack demo page, click **Logout**, then close the browser.

Step 4: Merging the Test Results

In this step, you merge the snapshots taken during Step 3: Running the Session, to get a complete picture of the coverage provided by these two test cases.

You merge the snapshots using the `jpcovmerge` command line tool.

To merge the snapshots:

1. Click **Start > Run** and type the following, then click **OK**:

   - In a Windows command window:
     ```
     <JPROBE_HOME>\bin\jpcovmerge
     <working_directory>\coverage_normal.jpc
     <working_directory>\coverage_loitering.jpc
     <working_directory>\coverage_merged.jpc
     ```
   - In a UNIX or Linux shell:
     ```
     <JPROBE_HOME>/bin/jpcovmerge
     <working_directory>/coverage_normal.jpc
     <working_directory>/coverage_loitering.jpc
     <working_directory>/coverage_merged.jpc
     ```

   Important This command fails if `<jprobe_home>` or `<working_directory>` has spaces in it. You must quote the application/arguments in this case.

2. On the JProbe Console, click **File > Open Snapshot**.
3. In the Open dialog box, select *<working_directory>\coverage_merged.jpc* and click **Open**.

   The merged snapshot is selected and its content is displayed in the Snapshot Browser view.

Step 5: Assessing Your Test Coverage

After merging your test cases, you are now ready to investigate your results. By default, JProbe displays results in terms of misses. A miss means that the code was not called during your test cases.

To investigate your results:

1. In the Snapshot Browser view, expand the class tree in the upper pane to show all of the classes in `demos.gamepack.minesweeper`, as this is the game that was exercised.
2. Click the **% Missed Methods** column heading in the upper pane to sort the table from highest-to-lowest percentage of missed methods.

Methods are missed in several classes: `MineSweeperException`, `MineSweeperModel`, `MineSweeperScore`, `MineSweeperSnapshot`, `MineSweeperMove`, and `MineSweeperRenderer`. We will investigate the first two classes.

3 In the upper pane, select the `MineSweeperModel` class.

The methods for this class are displayed in the lower pane.

4 Select the `play()` method, right-click, and select **Show Source** to open the *<JPROBE_HOME>\demos\gamepack\src\demos\gamepack\minesweeper\MineSweeperModel.java* source code.

The source code opens at a block of missed lines that are in an `if` statement, so you know that the test case missed a condition.

5 Click the **Down** button at the top of the coverage bar (to the right of the scroll bar) to move to the next set of missed lines.

6 Continue in this way until you have examined all the missed blocks of code, then move on to the next missed class.

Tip Once you identify an area for improvement, you can modify the test suite and redo the tutorial to see the improvement in overall coverage.

Creating a User Account for Game Pack

The first time that you start the Game Pack, you need to create a user account for yourself.

To create a user account:

1 In the Game Pack demo home page, click **Sign up as new user**.

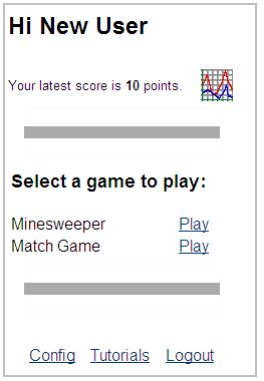
2 Type a user name and password in the **User ID** and **Password** fields.

3 Re-type the password in the **Re-enter Password** field.

4 Type the name you want to use for the games in the **Name** field.

5 Click **Sign-up**.

The Game Pack demo home page re-appears. The name you typed in the Name field appears in the top left corner.



You can now proceed to play Minesweeper or Match Game.

Game Pack Known Issues

The following known issues have been identified:

- The Game Pack may not work with the evaluation version of WebLogic Server. Use a fully licensed version of WebLogic or use JBoss.
- Minesweeper handles one selection at a time and does not buffer multiple tile clicks.
- Right-clicking Minesweeper and selecting **Open in New Window** may cause unexpected results.
- Opening multiple browser sessions for the same user may cause point calculation issues.

5

JProbe Treasurehunt

This section provides information about the Treasurehunt demo software that ships with JProbe.

This chapter contains the following sections:

Installing the Treasurehunt Demo 138
Using the Treasurehunt Demo 138

Installing the Treasurehunt Demo

For detailed installation instructions, see the *JProbe Installation Guide*.

If you chose to install examples during the JProbe installation, the Treasurehunt demo files are installed automatically in the following directory structure:

JPROBE_HOME

demos\

treasurehunt\

data\	(Input data required to run the treasurehunt demo.)
Maven\	(An example of how to integrate JProbe with Treasurehunt using the Maven build tool. These files give you a better understanding of how JProbe can be run as part of your build process, using Maven.)
src\	(Source code.)

Using the Treasurehunt Demo

The Treasurehunt demo application helps you understand how to use the JProbe Ant Tasks in your build file. For detailed information, see the Treasurehunt Tutorials in the *JProbe Ant Tasks User Manual*.

Index

A

about JProbe 8
about Quest Software 10
Adventure tutorial 59

C

code coverage tutorial 89, 129
contacting Quest 11
coverage analysis
  demos, summary of 58
  tutorials
    case effectiveness Java SE 59
    Java EE 89, 129
coverage effectiveness, Java SE tutorial 59
creating, a database 96
creating, Game Pack user accounts 134

D

database, creating 96
deadlock, Java SE tutorial 42
demos
  Java EE, Game Pack 67
  Java SE
    coverage analysis 57
    memory analysis 13
    performance analysis 39
    Treasurehunt 137
deploying, Game Pack demo on
  Apache Geronimo 95
  BEA WebLogic Server 10.3 103
  JBoss 101
  Oracle WebLogic Server 11g 106
Diner tutorial 42
documentation
  core 9
  feedback 10
  suite 9

E

environment variables, setting 69

G

Game Pack
  advanced evaluation 95
  building the demo 99
  Coverage Analysis tutorial 89, 128
  creating a database 96
  creating user accounts 134
  deploying demo 95
  getting started 68
  installing 68
  known issues 135
  Memory Analysis tutorials 71, 110
  Performance Analysis tutorial 83, 122
  quick evaluation 70
  running with JProbe 109
  setting environment variables 69
  system requirements 69

Geronimo
building the demo 99
creating a database 96

I

inefficient algorithm, Java SE tutorial 47
installing
Game Pack 68
Treasurehunt 138

J

Java coverage effectiveness 59
Java EE
code coverage tutorial 89, 129
loitering objects tutorial 71, 110
object cycling tutorial 77, 116
performance bottleneck tutorial 83, 122
Java SE
deadlock tutorial 42
inefficient algorithm tutorial 47
leak example tutorial 16
object over-allocation 28
JProbe community 12

L

LeakExample tutorial 16
Loitering Objects tutorial 71, 110

M

memory analysis
demos, summary of 14
tutorials
leak example Java SE 16
loitering objects Java EE 71, 110
object cycling Java EE 77, 116
object over-allocation Java SE 28
memory leak, Java SE tutorial 16

N

Network tutorial 28

O

Object Cycling, Java EE tutorial 77, 116
object over-allocation, Java SE tutorial 28

P

performance analysis
demos, summary of 40
tutorials
bottleneck Java EE 83, 122
deadlock Java SE 42
inefficient algorithm Java SE 47
Performance Bottleneck tutorial 83, 122
Polynomial tutorial 47

R

running Game Pack with JProbe 109

S

support 11

T

technical support 11
text conventions 10
Treasurehunt 137
tutorials
coverage
case effectiveness Java SE 59
Java EE 89, 129
memory
leak example Java SE 16
loitering objects Java EE 71, 110
object cycling Java EE 77, 116
object over-allocation Java SE 28
performance
bottleneck Java EE 83, 122

deadlock Java SE 42
inefficient algorithm Java SE 47

U

using, Treasurehunt 138